NEW STRATEGIES FOR HIGH PERFORMANCE VLSI PHYSICAL DESIGN

BY

HUA XIANG

B.S., Peking University, 1997
M.S., Peking University, 2000
M.S., University of Texas at Austin, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

NEW STRATEGIES FOR HIGH PERFORMANCE VLSI PHYSICAL DESIGN

Hua Xiang, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2004
Martin D. F. Wong, PhD. Adviser

Physical design plays an important role in connecting front-end design and back-end design in chip development. In this thesis, we solve several important problems in physical design of VLSI circuits.

Chapter 2 addresses a floorplan problem that considers floorplanning and bus planning simultaneously. We propose an efficient evaluation algorithm to transform a sequence pair to a floorplan with buses inserted. Then simulated annealing is used to search for an optimal or near optimal solution.

Chapter 3 addresses a wire planning problem with the bounded over-the-block constraint. Two exact polynomial-time algorithms are presented, and both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists.

Chapters 4 and 5 are based on a min-cost max-flow algorithm. In chapter 4, we present the first polynomial-time algorithm for simultaneous pin assignment and routing for all two-pin nets between one source block and all other blocks. In chapter 5, we propose a polynomial-time algorithm for integrated pin assignment and buffer insertion.

Chapters 6 and 7 address ECO problems. Chapter 6 presents two algorithms to resolve overlaps between power rails and signal wires which are introduced by power rail redesign. In chapter 7, we propose an algorithm to eliminate capacitive crosstalk violations.

To Father and Mother

ACKNOWLEDGMENTS

I would like to thank my adviser, Professor Martin D. F. Wong, for his constant guidance and invaluable support throughout my graduate study. I am greatly benefited from his deep insight in technical problems and valuable advice on my research.

I am also grateful to the other members of my committee — Professors Janak Patel, Lenny Pitt, and Josep Torrellas — for their interest in my work.

Also I would like to acknowledge my colleagues for their cooperation and many thoughtful technical discussions. I am grateful to all the wonderful friends I made during my graduate study. Their friendship made my graduate study much more productive and enjoyable.

Finally, I would like to express my special thanks to my parents for their love, encouragement, and understanding throughout my graduate study.

TABLE OF CONTENTS

LIST OF FIGURES x

LIST OF TABLES xv

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 BUS-DRIVEN FLOORPLANNING 6

2.1 Introduction 6

2.2 Preliminary 9

2.3 Problem Formulation 10

2.4 Bus Ordering via Sequence Pair 12

2.4.1 A necessary condition for one bus 12

2.4.2 Bus ordering between two buses 13

2.4.3 Multiple bus ordering 15

2.5 Evaluation Algorithm 24

2.5.1 Feasible_Bus_Checking_Orientation 25

2.5.2 Bus_Ordering 25

2.5.3 Modified_LCS_Computation 25

2.6 BDF Algorithm 30

2.6.1 Perturbation (Move) 30

2.6.2 Cost Function 30

2.7 Soft Block Adjustment 31

2.8 Experimental Results 32

2.9 Conclusion 35

CHAPTER 3 WIRE PLANNING WITH BOUNDED OVER-THE-BLOCK WIRE CONSTRAINTS 36
3.1 Introduction 36
3.2 Problem Formulation 39
3.3 WP Algorithm 40
3.3.1 WP-Path algorithm 41
3.3.2 WP-Split algorithm 47
3.3.3 Comparison 50
3.4 Experimental Results 51
3.5 Conclusion 53

CHAPTER 4 SIMULTANEOUS PIN ASSIGNMENT AND ROUTING . . . 54
4.1 Introduction 54
4.2 Problem Definition 57
4.3 The Algorithm 61
4.4 Applications 66
4.4.1 ECO 66
4.4.2 Improvement on any given solution 68
4.4.3 Multiple-pin nets 71
4.5 Experimental Results 74
4.6 Conclusion 78

CHAPTER 5 INTEGRATED PIN ASSIGNMENT AND BUFFER PLANNING 79
5.1 Introduction 79
5.2 Pin Assignment and Buffer Planning for One Source Block (PBO) 82
5.3 The Algorithm 84
5.4 Pin Assignment and Buffer Planning (PB) 89
5.5 Improvement with Node Clustering 94
5.6 Experimental Results 95

5.7 Conclusion 98

CHAPTER 6 ECO ALGORITHMS FOR REMOVING OVERLAPS BETWEEN POWER RAILS AND SIGNAL WIRES 99

6.1 Introduction 99

6.2 PSO (Power rail - Signal wire Overlap) Problem 105

6.3 FP-Range (Fixed-Pin-decided Range) 107

6.4 Consistency Graph 117

6.5 PSO-H Algorithm 120

6.6 PSO-G Algorithm 123

6.7 Experimental Results 129

6.8 Conclusion 130

CHAPTER 7 AN ECO ALGORITHM FOR ELIMINATING CROSSTALK VIOLATIONS 131

7.1 Introduction 131

7.2 Crosstalk Violation Elimination 133

7.3 Preliminaries 135

7.3.1 FP-Range 135

7.3.2 Crosstalk model 135

7.4 CVE Algorithm 136

7.4.1 FCVE algorithm 136

7.4.2 SCVE 141

7.5 Optimization 147

7.5.1 Node clustering 147

7.5.2 Edge omitting 148

7.6 Experimental Results 151

7.7 Conclusion 153

CHAPTER 8 CONCLUSION 154
8.1 Summary 154
8.2 Future Research 156

REFERENCES 158

APPENDIX 165

VITA 168

LIST OF FIGURES

FIGURE PAGE

1.1 The design flow of a chip. 2

2.1 Two floorplans have the same chip size. (a) Two buses u (A, C) and v (B, E, H) are assigned. (b) Neither of the buses can be assigned. 7

2.2 A feasible horizontal bus $u =< H, t, \{A, B, C\} >$. $y_{max} = y_c + h_c$, $y_{min} = y_b$, and $y_{max} - y_{min} \geq t$. 11

2.3 A necessary condition for one bus. (a) The sequence pair must be $(... A ... D ... B ... C ... , ... A ... D ... B ... C ...)$ to fit in a horizontal bus. (b) The sequence pair must be $(... B ... C ... A ..., ... A ... C ... B ...)$ to fit in a vertical bus. . . 13

2.4 Cases of relative positions of two horizontal buses. 16

2.5 Two kinds of cycles in bus ordering constraint graphs. (a) Two buses are crossing. (b) The bus ordering constraint graph corresponding to (a). (c) Three buses are crossing. (d) The bus ordering constraint graph corresponding to (c). 18

2.6 Independent set problem and node-deleting problem. (a) An instance of independent set problem (ISP). (b) G_d is a horizontal bus ordering constraint graph. 20

2.7 Node Deleting Algorithm. (a) An instance of bus ordering constraint graph G. (b) Nodes whose in-degree or out-degree is zero are removed from G. (c) Node c is deleted from G in order to break cycles. (d) The residual acyclic graph of G after deleting c and i. 22

2.8 Insert two horizontal buses to the floorplan represented by ($A\ D\ E\ B\ C\ F\ G$, $E\ D\ A\ F\ B\ C\ G$). (a) One horizontal bus $\{A, B, C\}$ is assigned. (b) In order to insert another bus $\{B, E, G\}$, blocks A and D have to move up and this makes the bus $\{A, B, C\}$ changed, too. 26
2.9 (a) Two buses overlap due to basic alignment adjustment. (b) Assignment of two buses without overlap. 27
2.10 (Case A) Two buses share blocks A and B. *Bus_Overlap* may happen. (Case B) The blocks of two buses appear interlaced along x-axis. *Bus_Overlap* may happen. (Case C) Two buses have no overlaps along x-axis. *Bus_Overlap* is impossible. 28
2.11 Soft block adjustment. (a) A BDF solution. Block E is on an LCS path. (b) The new BDF solution after changing the shape of block E. 31
2.12 The result packing of ami49-2 after soft block adjustment. There are 49 blocks and 12 buses. 33
2.13 An optimal packing of grid7. There are 49 blocks and 14 buses. 35

3.1 The routing illustrated by thin lines is not valid; while the routing shown by wide lines is a feasible solution. 38
3.2 (a) A routing-block B_i is divided into 3×3 subblocks, and the interconnect bound is 3. (b) A graph denotes all valid OB-wires within B_i. 41
3.3 (a) The two nodes r_i^j and r_i^k are within the same routing-block B_i. (b) Each node is split into an in-node and an out-node. 42
3.4 The solid lines indicate a routing solution between blocks B_1 and B_6. 44
3.5 The corresponding path graph G_p. The wide lines illustrate a shortest path from u_1^1 to u_6^1. 45
3.6 (a) r_i^1, r_i^2, r_i^3, and r_i^4 are subblocks of a routing-block B_i. (b) Each subblock is represented by a node array. 47
3.7 The corresponding split graph for Figure 3.4. The wide lines illustrate a shortest path from $v_1^1[1]$ to $v_6^1[1]$. 49

3.8 The comparison of WP-Path and WP-Split on the relationship between running time and the number of nets. 53

4.1 (a) The two-step approach fails to route all nets. (b) The optimal solution of pin assignment and routing by our approach. 54
4.2 (a) The net-by-net approach fails to route all nets. (b) The optimal solution of pin assignment and routing by our approach. 56
4.3 A routing grid graph for two layers. 58
4.4 (a) A PAR problem in detailed routing. (b) The corresponding network graph. 61
4.5 (a) A solution in a flow network. A flow f_1 goes from p to q; and another flow f_2 goes from q to p. (b) Another solution with less cost. 63
4.6 Node splitting for capacitated nodes. The capacity of the new edge is $U(r)$ and its cost is 0. 64
4.7 (a) A flow f in the network in Figure 4.4 (b), $|f| = 3$. (b) The corresponding solution of pin assignment and routing for the 3 nets in the problem of Figure 4.4 (a). 65
4.8 (a) The initial pin assignment and routing solution. (b) The solution obtained by applying PAR-by-Flow on Block A satisfying the new requirement. 67
4.9 Illustration of improvement on a given solution. (a) Illustration of net connections among Block A, B, C and D. (b) Initial net-by-net solution based on the min-cost path approach. 5 nets (2 between B and C; 3 between C and D) are not routed. The total cost is 51. (c) The solution after applying PAR-by-Flow on Block A. Cost is reduced by 15. (d) The solution after applying PAR-by-Flow on Block B. Two more nets between B and C are routed. (e) The solution after applying PAR-by-Flow on Block C. All nets are routed (3 more nets) with less cost (from 51 to 50). (f) The solution after applying PAR-by-Flow on Block D. Nothing is changed. 70

4.10 Illustration of improvement for a pin assignment and routing of two/multiple-pin nets. (a) A one-layer pin-assignment and routing solution. (b) When Block A is selected to be the source block, all nets connecting to A are removed to reroute. The routing e between B and C should not be changed. (c) The corresponding flow network graph. (d) A flow f ($|f| = 4$) in the network. . . . 72
4.11 An improved solution of pin assignment and routing of two/multiple-pin nets. . 73
4.12 Two-layer pin assignment and routing for X18. (a) Net-by-net solution. (b) The solution obtained by applying our method on (a). 75

5.1 (a) Three nets use 3 buffers and the total wire length is 19. (b) An optimal solution with 1 buffer and wire length 14. 80
5.2 (a) A PBO problem with 3 macro blocks and 3 buffer blocks. (b) The corresponding flow network graph. 83
5.3 Node splitting for capacitated nodes. The new edge has capacity $U(v)$ and cost $C(v)$. 86
5.4 (a) A flow f in the network in Figure 5.2 (b), $|f| = 3$. (b) The corresponding solution of pin assignment and buffer planning to the PBO problem of Figure 5.2 (a). 87
5.5 (a) A PB problem with 3 macro blocks and 3 buffer blocks. (b) The corresponding flow network when b_1 is the source block. 92
5.6 The corresponding flow network when b_2 is the source block. 93
5.7 The corresponding flow network of the PB problem in Figure 5.5(b) using the node clustering method. 94
5.8 (a) A flow f, $|f| = 3$ flows through a supernode. (b) The corresponding network and a flow solution. (c) Deriving connections for original pin nodes. . . . 95

6.1 (a) Some horizontal signal wire segments on M_5 overlap with P_1 and P_2. (b) A feasible PSO solution. (c) A solution with violations. 101
6.2 Wire separation requirement illustration. 106

6.3 (a) A PSO problem. (b) Overlaps: vertical segments a' and b' on M_4; and horizontal segments c and d on M_5. 108
6.4 FP-Range illustration. The tiny squares are fixed pins. 109
6.5 Three cases of vertical overlaps. 113
6.6 Illustration of the FP-Range calculation when the width is taken into consideration. 117
6.7 A routing solution of signal wires on the top layer 118
6.8 (a) Full connections of adjacent segments. (b) Consistency graph. 118
6.9 Illustration of Onodes/Rnodes. 124

7.1 (a) A routing solution with crosstalk violations. (b) A routing solution with overlap violations. 131
7.2 Segments A and C have capacitive crosstalk; while the crosstalk between segments B and C is zero. 136
7.3 (a) B, C, and D are three children of A. The position of A is fixed. (b) B is first selected and put to its highest available position. (c) A solution according to our approach. 139
7.4 (a) A CVEP problem. There are 4 signal wire segments A_1, A_2, A_3, and A_4, and 1 power rail P. (b) The consistency graph is a path. 141
7.5 (a) A CVEP problem. (b) FSP graph G of the CVEP problem. (c) SP graph $\bar{G}$ of the CVEP problem. (d) A feasible solution to the CVEP problem. 143
7.6 (a) A_i is a wire segment and it has 12 available positions. (b) Every three nodes are clustered as a "supernode". 147
7.7 (a) FSP graph of a CVEP problem. p is a feasible position of A_{i-1}. (b) SP graph of the CVEP problem. 148
7.8 (Case 1) u is the closest feasible position to y_i. Only one edge is needed. (Case 2) y_i is the only feasible position in $(B_u, 2y_i - u)$. Two edges are added. (Case 3) There are two feasible positions in $(2y_i - u, B_l)$. Three edges are added. . . 150

LIST OF TABLES

TABLE PAGE

2.1 Test set 1. 33

2.2 Test sets 2 and 3. 34

3.1 Algorithm comparison. 51

3.2 Test results of WP-Path and WP-Split algorithms. 52

4.1 Average results of 10 times for detailed routing test files. All nets are routed after refinement by RepIMProve-by-PAR. 76

4.2 Average results of 10 times for global routing test files. All nets are routed after refinement by RepIMProve-by-PAR. 77

5.1 Average results of PB-Flow for 5 times. All nets are found using PB-Flow algorithm. 97

6.1 Average results of PSO-H and PSO-G for 5 times. 129

7.1 Test files of CVE problem. 152

7.2 Test results of CVE problem. 152

7.3 Optimization for test file N3. 153

CHAPTER 1

INTRODUCTION

As the very large scale integration (VLSI) technology marches toward ultradeep submicron, designs are becoming increasingly complex with millions of layout objects on a monolithic chip. As a result, the chip design cycle time becomes longer. However, due to the rapid development of current technology and tight marketing schedule, the chip design time has to be short enough to satisfy time-to-market considerations. Therefore, sophisticated computer-aided design (CAD) tools and methodologies are badly needed, and they are widely used to facilitate chip development.

In general, the top-down design flow of a device can be abstracted as the following steps as illustrated in Figure 1.1 [1]. A design starts from specifications that describe the behavior of the target chip. Next, architectural design defines larger "circuit modules" such as arithmetic units, memory units, etc. Then the defined architecture is mapped to the logic structure in the logic design stage. Finally, physical design transforms the logic structure from the previous stage to geometric shapes that are used in the fabrication of the chip. Therefore, physical design plays an important role such that it connects front-end and back-end design. Moreover, the quality of physical design tools directly determines the performance and cost of the final product.

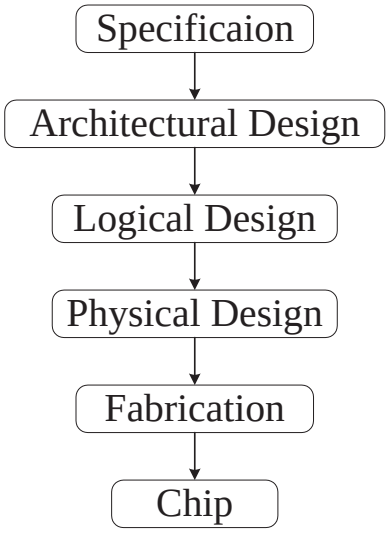


Figure 1.1 The design flow of a chip.

In this thesis, we study several important problems in physical design. In Chapter 2, we propose bus-driven floorplanning. In Chapter 3, we study the problem of wire planning with bounded over-the-block wires. In Chapter 4, we address the simultaneous pin-assignment and routing problem. In Chapter 5, we present an integrated approach for pin-assignment and buffer insertion. In Chapters 6 and 7, we propose algorithms to solve some ECO (engineering change order) problems. The results presented in these chapters can be briefly summarized as follows.

In Chapter 2, we propose an integrated approach for floorplanning and bus planning, i.e., bus-driven floorplanning (BDF). We are given a set of circuit blocks and bus specifications (i.e., the net list of blocks for the buses). A feasible BDF solution is a placement of all circuit blocks such that each bus can be realized as a rectangular strip (horizontal or vertical) going through all the blocks connected by the bus. The objective is to determine a feasible BDF solution that minimizes the floorplan area and the total bus area. Our ap-

proach is based upon the sequence-pair floorplan representation. After a careful analysis of the relationship between bus ordering and block ordering in the floorplan represented by a sequence pair, we derive feasibility conditions on sequence pairs that give feasible BDF solutions. We tested on three sets of test files and obtained excellent results.

In Chapter 3, we address the problem of wire planning (WP) with bounded over-the-block wires. The constraints on over-the-block wires help the longest over-the-block wires within a block to satisfy signal integrity without buffer inserted. We present two exact polynomial-time algorithms to solve the WP problem. Both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists. One requires less memory, while the other may take less running time when processing a large number of nets. According to different application requirements, users can choose an appropriate algorithm.

In Chapter 4, we present an algorithm for simultaneous pin-assignment and routing. In previous works, the algorithms for these problems can be classified into two categories: (1) a two-step approach where pin assignment is followed by routing, and (2) a net-by-net approach where pin assignment and routing for a single net are performed simultaneously. But none of the existing algorithms is "exact" in the sense that they may fail to route all nets even though a feasible solution exists. This remains to be true even if only two-pin nets with fixed pins between two blocks are concerned. In this chapter, we consider the problem of two-pin net connections from one macro block to all other blocks, and we present the first polynomial-time exact algorithm for simultaneous pin assignment and

routing for all two-pin nets between one block (source block) and all other blocks. In addition to finding a feasible solution whenever one exists, it guarantees to find a pin-assignment/routing solution with minimum cost $\alpha \cdot W + \beta \cdot V$, where W is the total wire length and V is the total number of vias. Our algorithm has various applications: (1) It is suitable in ECO situations where an existing solution is modified incrementally. (2) Given any pin assignment and routing solution obtained by any existing method, our algorithm can be used to increase the number of routed nets and reduce the routing cost. Furthermore, it provides an efficient algorithm for the pin assignment and routing problem of all blocks. The method is applicable to both global and detailed routing with arbitrary routing obstacles on multiple layers.

In Chapter 5, we present a polynomial-time exact algorithm for integrated pin assignment and buffer planning for all two-pin nets from one macro block (source block) to all other blocks. Moreover, we can guarantee to minimize the total cost $\alpha \cdot W + \beta \cdot R$ for any positive α and β where W is the total wire length and R is the number of buffers. By applying this algorithm iteratively (i.e., each time pick one block as the source block), it provides a polynomial-time algorithm for pin assignment and buffer planning for nets among multiple macro blocks. Experimental results demonstrate that this approach is efficient and effective.

In Chapter 6, we address the PSO (power rail - signal wire overlap) problem which removes overlaps between power rails and signal wires on the top layer of a multiple layer routing region under certain constraints. PSO problems are frequently caused by changes

from power delivery system or package design. Efficient and graceful solutions to PSO are needed due to design constraints and tight schedules during the late ECO stages. In this chapter, we first propose two algorithms to remove the overlaps between power rails and signal wires. Both algorithms guarantee to find a feasible solution as long as one exists. One is faster, while the other makes effort to minimize the total deviation as well as the maximum deviation. For a set of industrial test circuits, we were able to remove all overlaps between power rails and signal wires with minimal wire deviation.

In Chapter 7, we address the CVE (crosstalk violation elimination) problem. Due to the changes in a multiple layer routing design, the total capacitive crosstalk on some signal wire segments may be larger than their allowable bounds after post-layout timing/noise analysis. The target is to find a new routing solution without crosstalk violations under certain constraints which help to keep the new design close to the original one. We propose a two-stage algorithm to solve the CVE problem, and present optimization strategies to speed up the execution. One possible application of CVE algorithm is that it can be used to eliminate crosstalk violations in the output of PSO problem in Chapter 6.

Finally, in Chapter 8, we summarize our research and propose some directions for future research work.

CHAPTER 2

BUS-DRIVEN FLOORPLANNING

2.1 Introduction

As the deep submicron technology advances, chips become more congested even though more metal layers are used for routing. Usually a chip includes several buses. As design increases in complexity, bus routing becomes a heavy task, especially for networking chips or data processors. Since buses have different widths and go through several module blocks, the positions of macro blocks greatly affect bus planning. To ease bus routing and avoid unnecessary iterations in physical design, we need to consider bus planning in early floorplanning stage.

In this chapter, we address the problem of bus-driven floorplanning (BDF). We use top two layers for bus planning, and buses go either horizontally or vertically on one layer in floorplanning stage. The simple bus structure is good and efficient at planning stage, and would facilitate bus routing in later stages. Furthermore, more complicated bus structure can always be decomposed into several horizontal/vertical bus segments.

Informally, the problem can be described as follows. Given a set of rectangular macro blocks and the bus specifications (i.e., the net list of blocks for the buses), find a placement

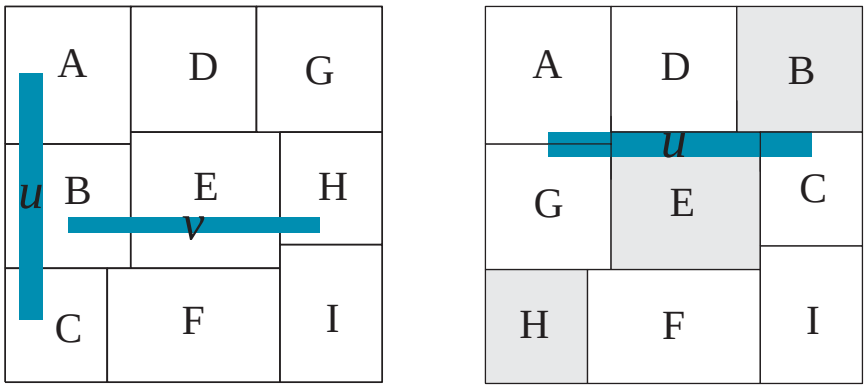


Figure 2.1 Two floorplans have the same chip size. (a) Two buses u (A, C) and v (B, E, H) are assigned. (b) Neither of the buses can be assigned.

of all circuit blocks such that each bus can be realized as a rectangular strip (horizontal or vertical) going through all the blocks connected by the bus. At the same time, the chip area as well as the total bus area is minimized.

Figure 2.1 gives an example. Figure 2.1(a) and (b) are two floorplans with the same chip size. Two buses u (A, C) and v (B, E, H) are placed in the floorplan of Figure 2.1(a). However, neither of the buses can be assigned based on the floorplan in Figure 2.1(b) since blocks B, E and H are not aligned, and the vertical overlap between blocks A and C is less than the width of bus u.

In previous works, researchers have discussed some particular kinds of floorplan constraints related to alignment. However, these kinds of alignment constraints are not suitable for bus-driven floorplanning. Young et al. [2] handle a kind of alignment in which modules involved in an alignment are required to be aligned by left (right/bottom/upper) side. But this is not necessary in BDF problems. For example, the bottom sides of blocks B, E and H are not aligned, but bus v still fits in the floorplan in Figure 2.1(a). Tang and

Wong [3] proposed another alignment constraint in which several blocks are aligned in a row, abutting with each other. But blocks involved in one bus do not need to be placed adjacent to each other. In Figure 2.1(a), A and C are not adjacent while bus u is assigned. Liu et al. [4] discussed predefined coordinate alignment constraint in which some blocks are to be placed along a predefined coordinate within a small region. In BDF, there are no constraints on coordinates. Rafiq et al. [5, 6] proposed bus-based integrated floorplanning. However, the bus defined in their works is composed of bundles of wires connecting only two blocks. Also bus assignment is accomplished by global routing.

Most floorplan algorithms use simulated annealing to search for an optimal solution. The implementation of the simulated annealing scheme depends on a floorplan representation where a neighbor solution is generated and examined by perturbing the representation (called 'move'). In this chapter, we use the sequence pair representation and analyze the relationship between bus ordering and sequence pair representation. Then a fast evaluation algorithm is proposed to transform a sequence pair representation to a floorplan with buses inserted. The algorithm first derives a bus ordering based on some necessary conditions. Then a modified longest common subsequence algorithm is applied to decide block positions as well as bus assignments. Moreover, we also develop an efficient algorithm to handle soft modules to further improve solution quality. Experimental results on three sets of test files (MCNC benchmarks, industry test files,and bus grid test files) demonstrate the effectiveness and efficiency of our approach.

The rest of the chapter is organized as follows. Section 2.2 provides background infor-

mation on the sequence pair representation. The formal definition of the BDF problem is given in Section 2.3. In Section 2.4, we analyze the relationship between bus ordering and sequence pair representation. Then a fast evaluation algorithm is proposed to transform a sequence pair to a BDF solution in Section 2.5. In Section 2.6, a simulated annealing BDF algorithm is presented. Finally, we address how to handle soft blocks to improve solution quality in Section 2.7. Experimental results are given in Section 2.8, and Section 2.9 concludes the paper.

2.2 Preliminary

A sequence pair is a pair of sequences of n elements representing a list of n blocks. In general, a sequence pair imposes the relationship between any two blocks a and b as follows:

(i) If a is ahead of b in both sequences, a is to the left of b in the floorplan.

(ii) If a is ahead of b in the first sequence while behind b in the second sequence, a is above b in the floorplan.

The original paper which proposed sequence pair [7] presented an algorithm to transform a sequence pair to a floorplan in $\Theta(n^2)$ time. Recently, Tang et al. sped up the evaluation algorithm to $O(n \log n)$ in [9], and later further to $O(n \log \log n)$ in [8].

The coordinates of blocks and the width and height of a floorplan can be obtained by computing longest common subsequence (LCS) in terms of the two sequences [9, 8]. Given a sequence pair (X, Y), the width of a floorplan equals the length of the longest

common subsequence of X and Y where weights are blocks' widths. Furthermore, given a block b, let $(X, Y) = (X_1 b X_2, Y_1 b Y_2)$ and $LCS(X, Y)$ be the length of the longest common subsequence of (X, Y). Then the x-coordinate of block b equals to $LCS(X_1, Y_1)$ with blocks' widths as weights. Similarly, the height of a floorplan is determined by dealing with the longest common subsequence of (X, Y^R) where Y^R is the reverse of Y and weights are blocks' heights. Furthermore, all the computations of blocks' x/y coordinates can be integrated into a single longest common subsequence computation for a sequence pair.

2.3 Problem Formulation

Suppose the routing region has multiple layers and buses can be assigned on the top two layers. So the orientation of buses is either horizontal or vertical. The problem of bus-driven floorplanning (BDF) can be defined as follows.

Problem 2.1 Bus-Driven Floorplanning (BDF) *Given n rectangular macro blocks $B = \{b_i | i = 1, ..., n\}$ and m buses $U = \{u_i | i = 1, ..., m\}$, each bus u_i has a width t_i and goes through a set of blocks B_i where $B_i \subseteq B$ and $|B_i| = k_i$. Decide the positions of macro blocks and buses such that there is no overlap between any two blocks or between any two horizontal (vertical) buses, and bus u_i goes through all of its k_i blocks. At the same time, the chip area as well as the total bus area is minimized.*

In BDF problems, buses should go through all of their related blocks. So the positions of blocks greatly affect bus assignments. For convenience, let $< g, t, \{b_1, ..., b_k\} >$ represent

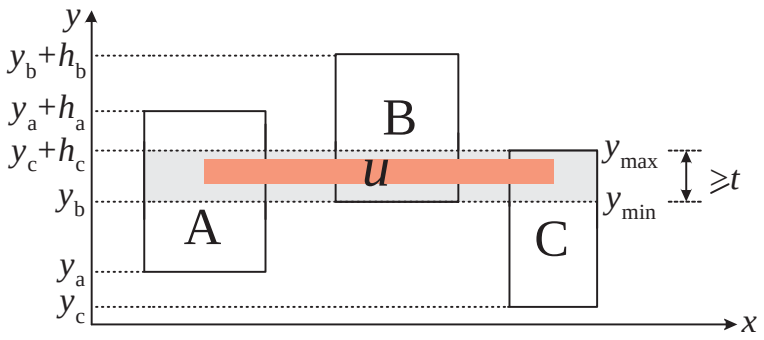


Figure 2.2 A feasible horizontal bus $u =< H, t, \{A, B, C\} >$. $y_{max} = y_c + h_c$, $y_{min} = y_b$, and $y_{max} - y_{min} \geq t$.

a bus u where $g \in \{H, V\}$ is the orientation, t is the bus width, and b_i ($i = 1, ..., k$) are the blocks the bus goes through. For short, a bus is just represented as $\{b_1, ..., b_k\}$. Also let (x_i, y_i) be the lower-left corner of block b_i. And the width and height of block b_i are w_i and h_i respectively. In the following, we give the necessary conditions of a feasible horizontal and vertical bus respectively.

Lemma 2.1 Feasible Horizontal Bus (H-Bus) *If a horizontal bus* $u =< H, t, \{b_1, ..., b_k\} >$ *is feasible, then* $y_{max} - y_{min} \geq t$ *where* $y_{max} = \min\{y_i + h_i | i = 1, 2, ..., k\}$ *and* $y_{min} = \max\{y_i | i = 1, 2, ..., k\}$.

Lemma 2.2 Feasible Vertical Bus (V-Bus) *If a vertical bus* $u =< V, t, \{b_1, ..., b_k\} >$ *is feasible, then* $x_{max} - x_{min} \geq t$ *where* $x_{max} = \min\{x_i + w_i | i = 1, 2, ..., k\}$ *and* $x_{min} = \max\{x_i | i = 1, 2, ..., k\}$.

Figure 2.2 illustrates an H-bus $u =< H, t, \{A, B, C\} >$. In order to fit in bus u, the vertical overlap of the three blocks has to be larger than the bus width t.

2.4 Bus Ordering via Sequence Pair

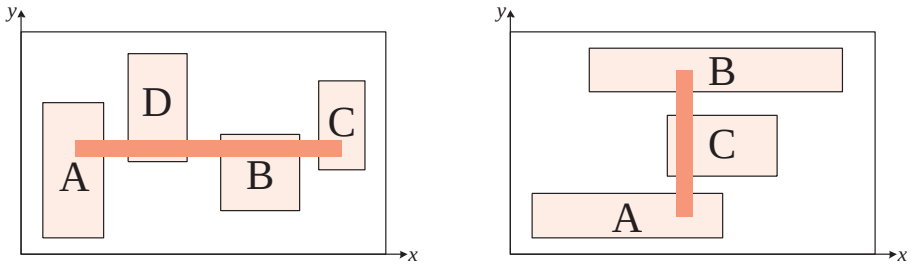
A sequence pair always entails a packing if no constraints are given. However, when constraints are introduced, there may not exist a corresponding packing for some sequence pairs.

In this section, we discuss the relationship between bus ordering and sequence pair representation. First, a necessary condition is derived when only one bus is considered. Then we discuss the relative positions of any two horizontal (vertical) buses imposed by a sequence pair. Based on the analysis of the ordering of two buses, we set up a bus ordering constraint graph and propose an algorithm to remove infeasible buses.

2.4.1 A necessary condition for one bus

Since blocks cannot overlap in a BDF solution, blocks have at most one-dimension overlap; i.e., if the projections on x-axis of two blocks have overlap, their projections on y-axis cannot overlap. On the other hand, if the projections on y-axis of two blocks have overlap, their projections on x-axis cannot overlap. However, in order to fit in a bus $\{b_1, ..., b_k\}$, the projections on x-axis (y-axis) of b_i and b_j ($i, j = 1, ..., k; i \neq j$) must have overlap. In other words, the position relationship of any two related blocks has to be left-right (below-above). Thus we have the following necessary condition.

Theorem 2.1 (Block Ordering) *Given a sequence pair (X, Y) and a bus $u = \{b_1, ..., b_k\}$, if u is feasible, then the ordering of the k blocks should be either the same or reverse in the*



(a) (b)

Figure 2.3 A necessary condition for one bus. (a) The sequence pair must be $(... A ... D ... B ... C ... , ... A ... D ... B ... C ...)$ to fit in a horizontal bus. (b) The sequence pair must be $(... B ... C ... A ..., ... A ... C ... B ...)$ to fit in a vertical bus.

two sequences X and Y. Furthermore, if the k blocks appear in the same order in both X and Y, the orientation of the bus is horizontal; otherwise the bus is vertical.

For convenience, this necessary condition is also called *block ordering*. Figure 2.3 gives two examples. The sequence pair for Figure 2.3(a) is $(... A ... D ... B ... C ... , ... A ... D ... B ... C ...)$, and a horizontal bus $\{A, B, C, D\}$ can be assigned. Figure 2.3(b) shows another example. The sequence pair is $(... B ... C ... A ..., ... A ... C ... B ...)$ and the bus is a vertical one $\{A, B, C\}$. Note that Theorem 2.1 deals with only one bus. When multiple buses are considered, it is likely that some buses cannot be assigned for the floorplan although each bus satisfies the necessary condition.

2.4.2 Bus ordering between two buses

The relative positions of blocks is determined by a sequence pair. Since buses go through blocks, the ordering of buses is also influenced by the sequence pair.

Given a sequence pair (X, Y) and two horizontal buses $u = \{a_1, a_2, ..., a_k\}$ and $v =$

$\{b_1, b_2, ..., b_l\}$, denote the block set $S_u = \{a_1, a_2, ..., a_k\}$, $S_v = \{b_1, b_2, ..., b_l\}$, and $S = S_u \cup S_v$. Suppose $|S| = L$ ($L \leq k + l$ since the two buses may go through the same blocks) and (X, Y) satisfies *block ordering* for the two buses. Also we assume these L blocks appear in the sequence pair as $(... \ c_1 \ ... \ c_2 \ ... \ ... \ c_L \ ... \ , \ ... \ d_1 \ ... \ d_2 \ ... \ ... \ d_L \ ...)$ where $c_i \in S$ and $d_i \in S$ $(i = 1, ..., L)$, and the subsequence pair $(X', Y') = (c_1 \ c_2 \ ... \ c_L \ , \ d_1 \ d_2 \ ... \ d_L)$. For convenience, let $p[c_i] = i$ $(i = 1, ..., L)$ which denotes the position of c_i in X', and $q[d_i] = i$ represents the position of d_i in Y'. From this subsequence pair (X', Y'), we can derive the relative positions of the two buses.

Case 1. If $\forall a \in S_u$, $p[a] \geq q[a]$, and $\exists a \in S_u$, $p[a] > q[a]$, then bus u is below bus v.

Suppose $p[a_i] > q[a_i]$, then (X, Y) must be $(... \ b_j \ ... \ a_i \ ... \ , \ \ ... \ a_i \ ... \ b_j \ ...)$. b_j is above a_i. Since u goes through a_i while v goes through b_j, bus u is below v.

Figure 2.4 (Case 1) shows an example. The subsequence pair is ($D \ A \ E \ B \ F \ C$, $A \ D \ B \ E \ C \ F$). $p[A] = 2$ and $q[A] = 1$; $p[B] = 4$ and $q[B] = 3$; $p[C] = 6$ and $q[C] = 5$. So bus $u = \{A, B, C\}$ is below bus $v = \{D, E, F\}$.

Case 2. If $\forall a \in S_u$, $p[a] \leq q[a]$, and $\exists a \in S_u$, $p[a] < q[a]$, then bus u is above bus v.

Figure 2.4 (Case 2) shows an example. Block B is shared by both buses. Bus $u = \{A, B, C\}$ is above bus $v = \{D, B, E\}$.

Case 3. If $\exists a \in S_u$, $p[a] > q[a]$, and $\exists a' \in S_u$, $p[a'] < q[a']$, then the two buses u and

v cannot be assigned at the same time.

Suppose $p[a_i] > q[a_i]$ and $p[a_j] < q[a_j]$, then (X, Y) must be $(... \, b_I \, ... \, a_i \, ... \, a_j \, ... \, b_J \, ...$, $... \, a_i \, ... \, b_I \, ... \, b_J \, ... \, a_j \, ...)$. Block b_I is above a_i while b_J is below a_j. The positions of blocks are illustrated in Figure 2.4 (Case 3).

In the example, the two buses are $u = \{A, B\}$ and $v = \{C, D\}$. Then the subsequence pair is (X', Y') is $(A \; C \; D \; B, \; C \; A \; B \; D)$. For block A, $p[A] = 1$ and $q[A] = 2$ while $p[B] = 4$ and $q[B] = 3$. In this case, the two buses cannot be assigned at the same time.

Case 4. If $\forall a \in S_u$, $p[a] = q[a]$, then the two buses have no firm ordering. Either bus can be above the other.

Figure 2.4 (Case 4) illustrates an example. In this example, bus $u = \{A, B, C\}$ can be below bus $v = \{D, E\}$. On the other hand, u is also possible above v. Therefore, the two buses have no bus ordering constraints.

For any two vertical buses, we can get the similar results from (X, Y^R).

2.4.3 Multiple bus ordering

In a BDF solution, it is impossible that the ordering of several buses forms a cycle. For example, bus u is above bus v, bus v is above bus w, and bus w is above bus u. This kind of relationship cannot exist in a feasible solution.

In the above section, we have discussed bus ordering imposed by the given sequence pair. To express the relative positions among buses, we construct bus ordering constraint

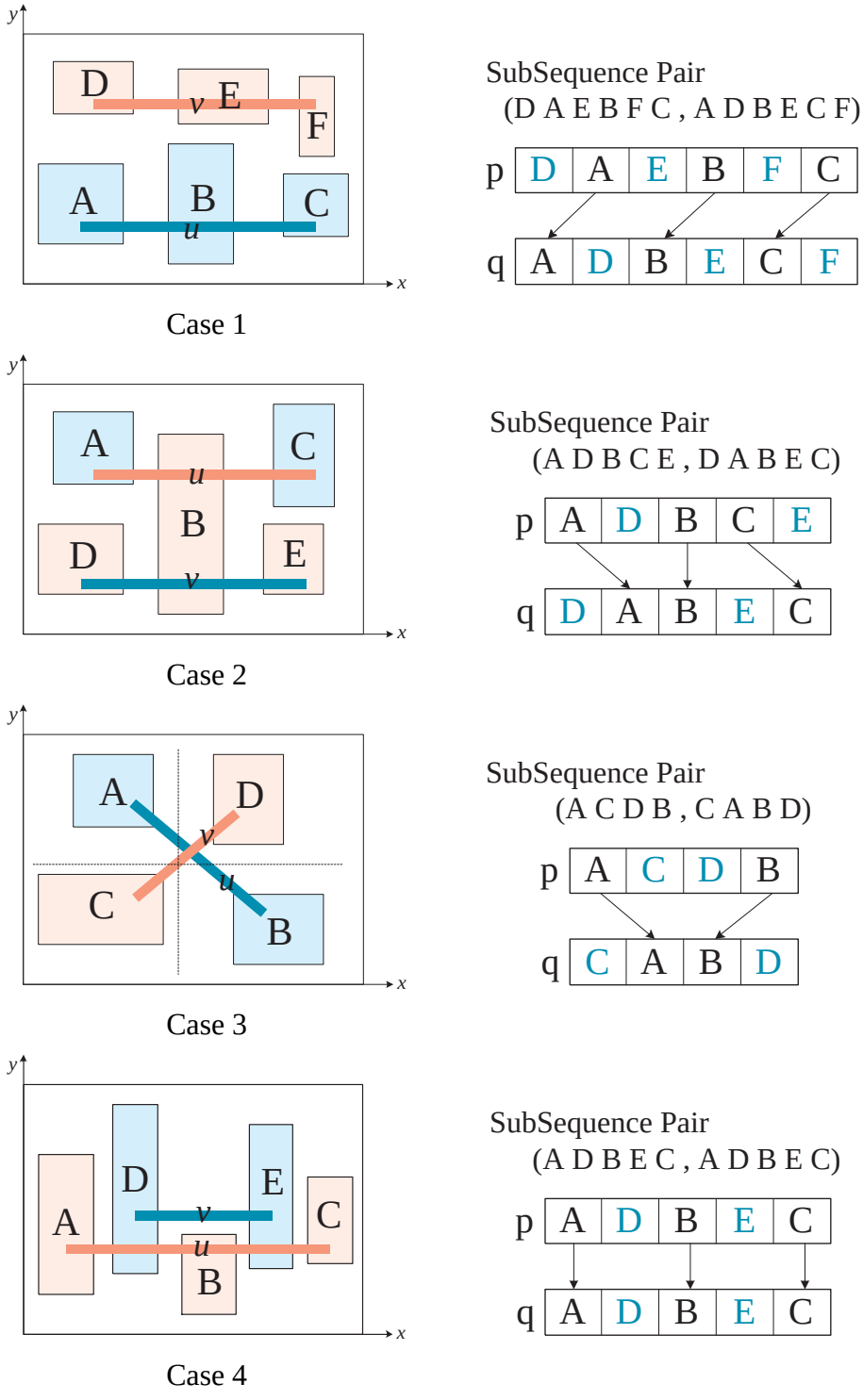


Figure 2.4 Cases of relative positions of two horizontal buses.

graphs for horizontal buses and vertical buses, respectively. The construction rules for a horizontal bus ordering constraint graph are listed as follows. The graph for vertical buses can be derived similarly.

- Each bus is represented by a node.
- If one bus u is above another bus v (Case 1 or 2), add one edge (u, v).
- If one block related to bus u is above a block related to bus v, while another block related to u is below a block related to v (Case 3), add two edges (u, v) and (v, u).
- If two buses have no bus ordering constraint (Case 4), no edge is added.

The horizontal bus ordering constraint graph serves in two ways:

(i) Given a BDF solution, the block packing must correspond to a sequence pair. Then the horizontal bus relationship imposed by the sequence pair can be represented by an acyclic constraint graph.

(ii) If a constraint graph contains a cycle, then at least one bus cannot be assigned. According to the construction rules, there are two kinds of cycles.

  (a) A cycle includes only two nodes. Then the relative position of the two corresponding buses must comply with Case 3, and at least one bus cannot be assigned. Figure 2.5(a) shows an example. Two buses $u = \{A, B\}$ and $v = \{C, D\}$ are crossing, and the subgraph is given in Figure 2.5(b).

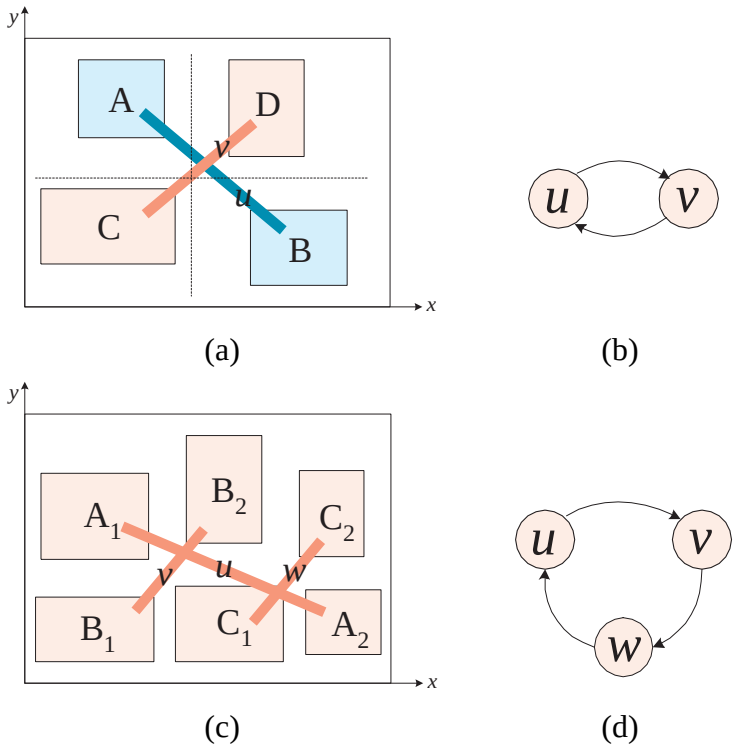


Figure 2.5 Two kinds of cycles in bus ordering constraint graphs. (a) Two buses are crossing. (b) The bus ordering constraint graph corresponding to (a). (c) Three buses are crossing. (d) The bus ordering constraint graph corresponding to (c).

(b) A cycle includes at least three nodes. Figure 2.5 (c) illustrates an example. There are three buses $u = \{A_1, A_2\}$, $v = \{B_1, B_2\}$ and $w = \{C_1, C_2\}$. The sequence pair is $(... \ A_1 \ ... \ B_1 \ ... \ B_2 \ ... \ C_1 \ ... \ C_2 \ ... \ A_2 \ ... \ , ... \ B_1 \ ... \ A_1 \ ... \ C_1 \ ... \ B_2 \ ... \ A_2 \ ... \ C_2 \ ...)$. From this sequence pair, we can conclude that bus u should be above v, v should be above w, and w should be above u. However, this is impossible in a BDF solution. Therefore, at least one bus has to be discarded.

If a bus-ordering constraint graph contains cycles, there must be some buses that cannot be assigned. Since our target is to assign as many buses as possible, the problem becomes how to remove minimum number of buses so that the graph is acyclic. However this prob-

lem is an NP-Complete problem.

For convenience, if some nodes are removed from the graph $G = (V, E)$, then edges connecting to/from these nodes are also removed, and the result graph is called a residual graph. Also all nodes are indexed, and node $u < v$ means that the index of u is less than that of v.

Problem 2.2 Node-Deleting Problem (NDP) *Given a sequence pair and a set of buses, a horizontal (vertical) bus-ordering constraint graph can be constructed. Remove nodes from the constraint graph so that the residual graph is acyclic. At the same time, the number of deleted nodes is minimized.*

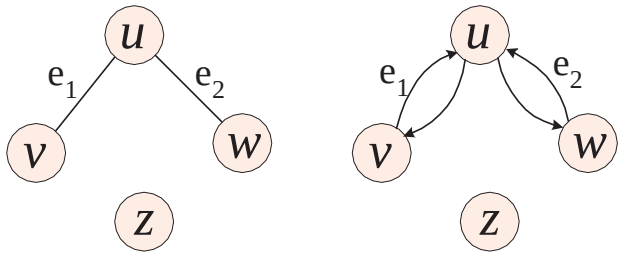
Theorem 2.2 *Node-Deleting Problem (NDP) is NP-Complete.*

Proof

NDP is the optimization problem of removing minimum number of nodes from a constraint graph so that the residual graph is acyclic. As a decision problem, we ask simply whether there exists a node set of size k such that the residual graph is acyclic by removing these nodes from a constraint graph.

To show that NDP $\in$ NP, for a given constraint graph $\bar{G} = (\bar{V}, \bar{E})$, we use the set $\bar{V}' \subseteq \bar{V}$ as a certificate of $\bar{G}$. Checking whether the residual graph of removing $\bar{V}'$ from $\bar{G}$ is acyclic or not can be accomplished in polynomial time.

Next we prove that NDP is NP-hard by proving that independent set problem (ISP), which is NP-complete, is polynomial-time reducible to NDP; i.e., ISP $\leq_p$ NDP.



(a) (b)

Figure 2.6 Independent set problem and node-deleting problem. (a) An instance of independent set problem (ISP). (b) G_d is a horizontal bus ordering constraint graph.

Let $G = (V, E)$ be an instance of ISP. Suppose $|V| = N$ and $|E| = M$. We form a directed graph $G_d = (V, E_d)$ where $E_d = \{(i,j)|(i,j) \in E\} \cup \{(j,i)|(i,j) \in E\}$; i.e., each edge in G is represented by a pair of edges with different directions in G_d. Obviously, the construction of G_d from G takes $O(M)$ running time. Figure 2.6 shows an example. Figure 2.6(a) is an instance of independent set problem G. Figure 2.6(b) is G_d. Note that G is an undirected graph.

Given a node subset $\bar{V}$, we can get a residual graph $\bar{G}_d$ of G_d by removing nodes in $\{V - \bar{V}\}$. We show that $\bar{V}$ is an independent set of G if and only if the residual graph $\bar{G}_d$ is acyclic. If $\bar{V}$ is an independent set of G, then there are no edges in $\bar{G}_d$. Obviously, $\bar{G}_d$ is acyclic. On the other hand, for any residual graph $\bar{G}_d$ of G_d, if it contains no cycles, then there are no edges in $\bar{G}_d$ since edges always appear in pairs. Therefore, the nodes in $\bar{G}_d$ also form an independent set of G.

Finally we show that G_d is a bus ordering constraint graph.

For any edge $e_i = (u, v) \in E$ **(**$u < v$**)**, let block sequence $x_i = (a_i^u\ b_i^v\ c_i^v\ d_i^u)$, and block sequence $y_i = (b_i^v\ a_i^u\ d_i^u\ c_i^v)$, where a_i^u, b_i^v, c_i^v and d_i^u are macro blocks. Suppose there are

L independent nodes w_i ($i = 1, ..., L$) in G which are not incident on any edge. Let block sequence $x_{M+i} = (a^w_{M+i}\ d^w_{M+i})$ and block sequence $y_{M+i} = (a^w_{M+i}\ d^w_{M+i})$ where a^w_{M+i} and d^w_{M+i} are blocks.

We form a sequence pair $(X, Y) = (x_1...x_M x_{M+1}...x_{M+L}\ ,\ y_1...y_M\ y_{M+1}...y_{M+L})$. Blocks in X and Y form the macro block set B. So totally there are $4M + 2L$ blocks. Since there are N nodes in G, the number of buses is also N. For each bus p, the blocks that p goes through are $\{z^p_i | z^p_i \in B, i = 1, ..., (M + L), z = a, b, c, d\}$. Since the ordering of blocks of each bus is always the same in both sequences, all buses are horizontal buses.

For each pair of buses u and v ($u < v$), we get the subsequence pair $(X', Y') = (x'_1 x'_2 ... x'_M, y'_1 y'_2 ... y'_M)$, where x'_i is a subsequence of x_i, and y'_i is a subsequence of y_i. Blocks appearing in X' or Y' are related to either bus u or v. Furthermore, (x'_i, y'_i) $(i = 1...M)$ can be only one of the two cases.

(i) $(x'_i, y'_i) = (x_i, y_i)$

(ii) $x'_i = y'_i$

If $\exists J$, $(x'_J, y'_J) = (x_J, y_J)$, then J is unique since there is only one edge between two nodes u and v in G. At the same time, the subsequence pair (X', Y') involves bus crossing (Case 3) for bus u and v. Therefore, the bus constraint graph contains two edges (u, v) and (v, u).

On the other hand, if $\forall i \in \{1, ..., (M+L)\}$, $x'_i = y'_i$, the two buses have no bus ordering constraint, and there is no edge between the two nodes u and v in the constraint graph. Also if $x'_i = y'_i$ (x'_i/y'_i can be empty), then there is no edge between u and v in G either. Thus

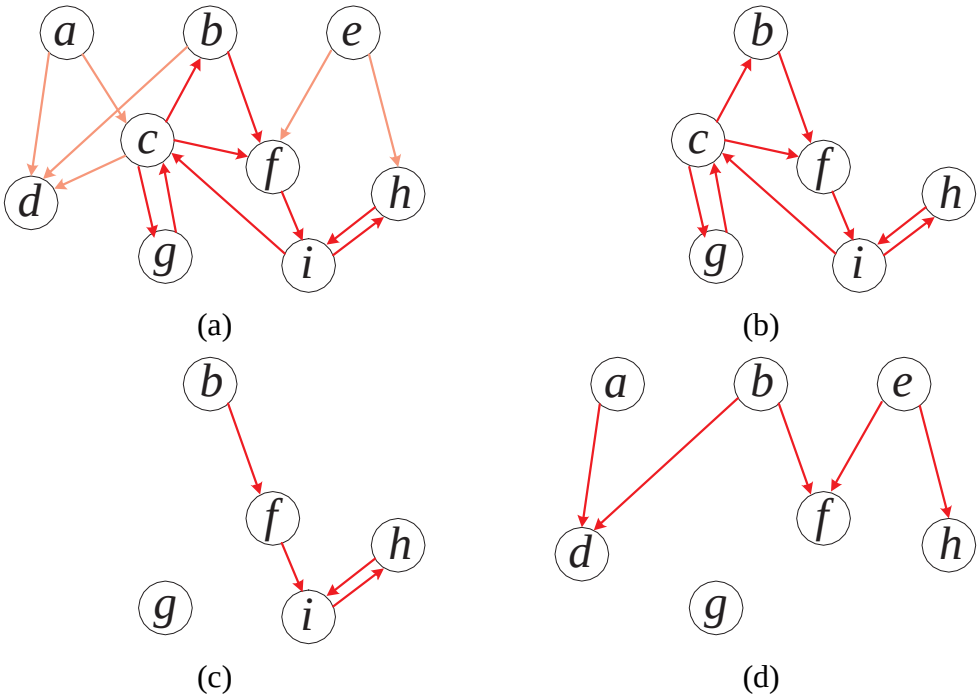


Figure 2.7 Node Deleting Algorithm. (a) An instance of bus ordering constraint graph G. (b) Nodes whose in-degree or out-degree is zero are removed from G. (c) Node c is deleted from G in order to break cycles. (d) The residual acyclic graph of G after deleting c and i.

we can conclude that G_d is the horizontal bus ordering constraint graph for the constructed sequence pair. ¶

Since NDP is NP-Complete, we derive a heuristic method to remove nodes from a graph so that the residual graph is acyclic. The method is based on the following lemma.

Lemma 2.3 *Given a directed graph, if the in-degree and out-degree for each node are both nonzero, then a cycle must exist in the graph.*

For any given directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$, if the in-degree and out-degree of each node are nonzero, a cycle can be found in the following way. Randomly select a node $\tilde{v}_1$ as the first node of a path. Then let the second node $\tilde{v}_2$ be a node incident on an out-going edge

of $\tilde{v}_1$. Since the out-degree of $\tilde{v}_1$ is nonzero, an out-going edge always exists. Repeat this process until a node appears twice along the path. Since $|\tilde{V}|$ is finite, the loop must finish within $|\tilde{V}| + 1$ steps. And if a node appears twice on a path, it means a cycle is formed.

An algorithm is used to remove nodes to break cycles in a bus-ordering constraint graph.

Algorithm 1 Node_Deleting (V, E)

```
1:  for i = 1 to |V| do
2:      Calculate in-degree and out-degree of nodes in V
3:      Find min in-degree min_in and min out-degree min_out
4:      if (min_in = 0) or (min_out = 0) then
5:          Remove the corresponding node v from V
6:          Remove edges connecting to/from v from E
7:      else
8:          Find the node v with max degree
9:          Insert v into Remove_Set
10:         Remove v from V and related edges from E
11:     end if
12: end for
13: return Remove_Set
```

For each iteration, the size of V is reduced by one. If we can find a node whose in-degree or out-degree is 0, then this node is treated as a good node. Otherwise, we select the node v with max degree (in-degree + out-degree) and insert it to the Remove_Set, i.e., v should be discarded in order to break cycles. This algorithm guarantees that if the graph is acyclic, Remove_Set is empty. The running time is $O(|V|^2)$.

Figure 2.7 illustrates an example. Figure 2.7(a) shows an instance of bus ordering constraint graph G, which includes nine buses. We first remove nodes whose in-degree

or out-degree is zero. Buses a, d, and e are removed from G as Figure 2.7(b). In Figure 2.7(b), the in-degree and out-degree of all nodes are nonzero; therefore, a cycle must exist. Since c has the maximum degree, c is deleted, making b and g free. The result is illustrated as Figure 2.7(c). Finally i is deleted to break the cycle between i and h. Therefore, there are two nodes c and i in Remove_Set. Figure 2.7(d) shows the residual acyclic graph after deleting c and i. Furthermore, based on Figure 2.7(d), it is easy to find a bus ordering consistent with the below-above relationship imposed by the sequence pair. For instance, the bus ordering (from bottom to top) could be g, d, f, h, a, b, e.

2.5 Evaluation Algorithm

The evaluation algorithm *Algorithm* 2 transforms a sequence pair representation to a BDF solution. However, for some sequence pairs, it is impossible to fit in all of the buses. For example, if a sequence pair violates *block ordering* for a bus, then some buses cannot be assigned. Therefore, the target of the evaluation algorithm is to find a floorplan that assigns as many buses as possible. The algorithm is summarized as follows. Suppose there are n blocks and m buses.

Algorithm 2 Evaluation_BDF (Seq, Bus)

1: Feasible_Bus_Checking_Orientation
2: Bus_Ordering
3: Modified_LCS_Computation

In the following, we explain the above three procedures one by one.

2.5.1 Feasible_Bus_Checking_Orientation

According to Theorem 2.1, if the blocks of a bus violate *block ordering* in the given sequence pair, the bus cannot be assigned. Therefore, the first step is to identify these buses and remove them from the bus set. For each bus, one scan of the sequence pair is enough to make the judgment. At the same time, if blocks related to a bus appear in the same order in both sequences, the bus is a horizontal bus, otherwise, the bus is a vertical one. This step takes $O(mn)$ time.

2.5.2 Bus_Ordering

Due to the bus ordering imposed by the given sequence pair, some buses cannot be assigned at the same time as discussed in Section 2.4. We apply the Node_Deleting algorithm to further remove some buses. Meanwhile, since the constraint graph is acyclic, we sort the horizontal buses from bottom to top (from left to right for vertical buses) according to the below-above (left-right) relationship. This bus order will be used in the next step. This step takes $O(m^2n)$ time.

2.5.3 Modified_LCS_Computation

The algorithm is based on the engine of computing longest common subsequence (LCS) presented in [8]. LCS computation calculates x coordinates and y coordinates separately. And it always packs blocks from bottom to top (from left to right). In this section, we only discuss the calculation of y coordinates of blocks with the assignment of horizontal buses.

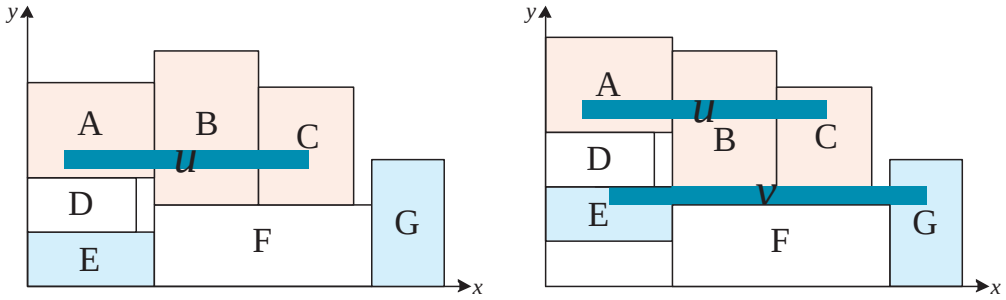


Figure 2.8 Insert two horizontal buses to the floorplan represented by $(A\ D\ E\ B\ C\ F\ G,\ E\ D\ A\ F\ B\ C\ G)$. (a) One horizontal bus $\{A, B, C\}$ is assigned. (b) In order to insert another bus $\{B, E, G\}$, blocks A and D have to move up and this makes the bus $\{A, B, C\}$ changed, too.

The calculation of x coordinates of blocks and vertical buses can be derived similarly.

For any given horizontal bus $< H, t, \{b_1, ..., b_k\} >$, the y coordinates of the k blocks are first calculated with LCS computation. Then these k blocks are aligned so that the bus can be inserted. Suppose the height of b_i is h_i and the y-coordinate of the lower-left corner of b_i is y_i. The basic alignment can be performed as follows:

$$y_{max} = \max\{y_i | i = 1, 2, ..., k\}$$

$$y_i = \max\{y_i, y_{max} + t - h_i\} \qquad \forall i \in \{i = 1, 2, ..., k\}.$$

However, the alignment adjustment for different buses may affect each other. For example, in Figure 2.8, the given sequence pair is $(A\ D\ E\ B\ C\ F\ G, E\ D\ A\ F\ B\ C\ G)$, and two buses $u = \{A, B, C\}$ and $v = \{B, E, G\}$ are to be inserted. Suppose the horizontal bus $u = \{A, B, C\}$ is first assigned as Figure 2.8 (a). When we want to place another bus $v = \{B, E, G\}$, block E has to move up, which causes blocks A and D to move up too. Furthermore, due to the change of block A, bus u has to be reassigned. Since LCS computation packs blocks from the bottom up, it is important to process buses in the same way.

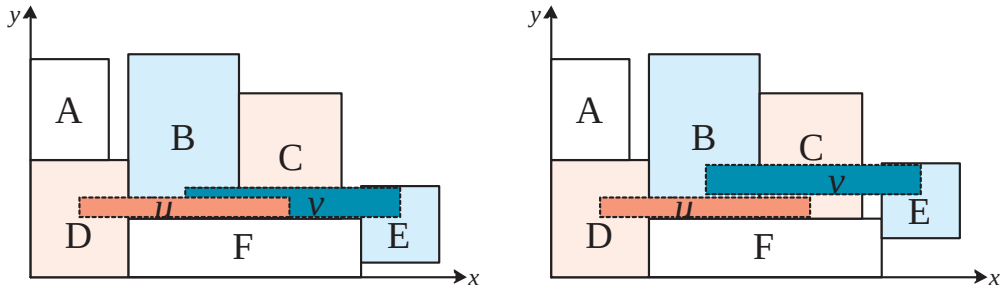


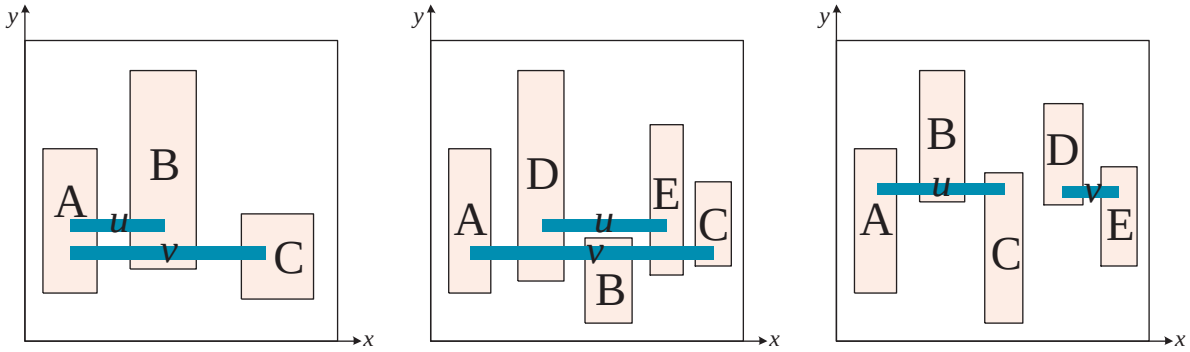
Figure 2.9 (a) Two buses overlap due to basic alignment adjustment. (b) Assignment of two buses without overlap.

Each time, the lowest bus is selected and assigned so that the bus would not be affected by later processing of other buses. By calling Bus_Ordering procedure, a sorted bus list can be obtained.

At the same time, when multiple horizontal buses are considered, we also need to avoid overlap between horizontal buses. Therefore, the above basic alignment is not enough.

For example, in Figure 2.9, two horizontal buses $u = \{C, D\}$ and $v = \{B, E\}$ are to be assigned. First, bus u is assigned with y_u as the y-coordinate of its bottom edge. However, according to the basic alignment calculation, the y-coordinate of bus v's bottom edge is also y_u. Then bus u and v are overlapped as Figure 2.9(a). This is not allowed in a BDF solution. We call this situation *Bus_Overlap*. Figure 2.9(b) shows a feasible solution.

If two buses have bus ordering constraint (only Case 1 and 2 in Section 2.4.2 are considered since only one bus can be assigned in Case 3), the above situation cannot happen. This is because in Case 1 or 2, there must exist a block of one bus which is above a block of the other bus. Then the two buses cannot overlap. On the other hand, if two buses have no bus ordering constraint (Case 4 in Section 2.4.2), *Bus_Overlap* may happen.



(Case A) (Case B) (Case C)

Figure 2.10 (Case A) Two buses share blocks A and B. *Bus_Overlap* may happen. (Case B) The blocks of two buses appear interlaced along x-axis. *Bus_Overlap* may happen. (Case C) Two buses have no overlaps along x-axis. *Bus_Overlap* is impossible.

Suppose two horizontal buses $u = \{a_1, a_2, ..., a_k\}$ and $v = \{b_1, b_2, ..., b_l\}$ have no bus ordering constraints. Denote the block set $S_u = \{a_1, a_2, ..., a_k\}$, $S_v = \{b_1, b_2, ..., b_l\}$. There are still three cases:

Case A: $S_u \cap S_v \neq \phi$. Two buses share at least one block. *Bus_Overlap* may happen, like Figure 2.10 (Case A) which involves two buses $u = \{A, B\}$ and $v = \{A, B, C\}$.

Case B: $S_u \cap S_v = \phi$, and the subsequence pair (X', Y') is $(... \, a_{i_1} \, ... b_j \, ... \, a_{i_k} \, ... \, , ... \, a_{i_1} \, ... \, b_j \, ... \, a_{i_k} \, ...)$. Figure 2.10 (Case B) shows an example. In this case, the blocks of two buses $u = \{A, B, C\}$ and $v = \{D, E\}$ appear interlaced along x-axis. It is likely that *Bus_Overlap* happens.

Case C: $S_u \cap S_v = \phi$, and in the subsequence pair (X', Y'), all blocks in S_u appear ahead of (behind) blocks in S_v. Figure 2.10 (Case C) shows an example. In this case, the

two buses $u = \{A, B, C\}$ and $v = \{D, E\}$ do not have overlap along x-axis. Therefore, the y-coordinates of the two buses can be decided independently; i.e., *Bus_Overlap* cannot happen.

Based on the above discussion, we can conclude that only Case A and B can lead to *Bus_Overlap*. Therefore, in the basic alignment calculation, we also need to check Case A and B so that no overlaps between two buses are introduced. This kind of checking can be incorporated in the Bus_Ordering procedure, and the results are kept in a table. If Case A or B is detected, the current bus has to move up until there is no overlap with the buses below.

In summary, after we get the bus list from Bus_Ordering procedure, we apply LCS computation m times (suppose there are m buses in the bus list). In one iteration, after the positions of blocks related to bus u are calculated by LCS, the position of bus u is first calculated by basic alignment calculation. Then we check if bus u has Case A or B with buses below u from the table which is created during Bus_Ordering. If there is this kind of buses, get the highest position of those buses and the position of bus u is decided by adding the bus width to the value. After the position of bus u is decided, check all of its related blocks, and move up blocks if necessary in order to let the bus go through. Once the position of a bus is calculated, the bus is not changed any more. Therefore, each iteration fixes one bus. The running time of LCS is $O(n \log \log n)$ [8] where n is the number of blocks. So Modified_LCS_Computation is bounded by $O(mn \log \log n + m^2 n)$.

2.6 BDF Algorithm

Most floorplan algorithms based on the sequence pair representation use simulated annealing (SA). In this paper, we also use SA to search an optimal or near optimal solution to a BDF problem.

2.6.1 Perturbation (Move)

We use the following operations to generate a neighboring sequence pair in simulated annealing:

1. *Swap* is to swap two blocks in either the first sequence or the second sequence. Swap can be done in constant time.

2. *Rotation* is to rotate a block (e.g. exchange the width and height of a block). Rotation does not cause any changes to the sequence pair. This operation can be done in constant time.

2.6.2 Cost Function

The target of BDF problem is to minimize the chip area and the total bus area. At the same time, we hope to insert all of the buses. Therefore, we define the cost function as follows:

$$Cost = \alpha \cdot C + \beta \cdot B + \gamma \cdot M$$

where C is the chip area, B is the bus area, M is the number of unassigned buses, and α, β, and γ are coefficients defined by users.

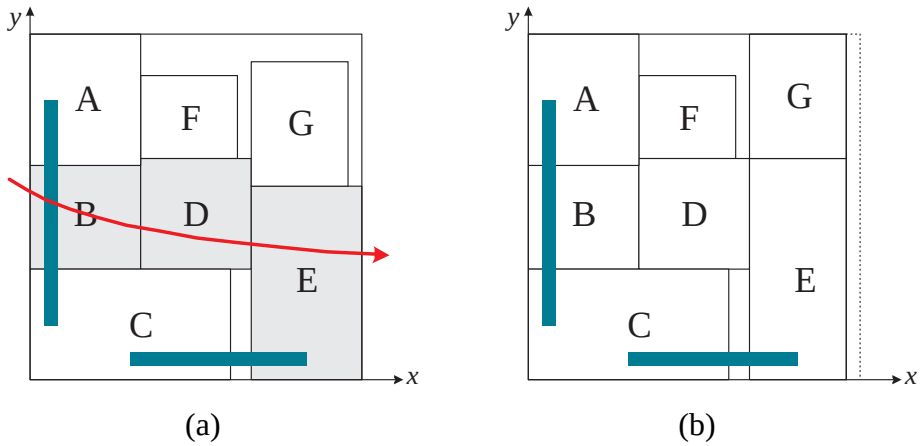


Figure 2.11 Soft block adjustment. (a) A BDF solution. Block E is on an LCS path. (b) The new BDF solution after changing the shape of block E.

By applying the Evaluation_BDF algorithm, the positions of blocks and buses are all calculated. Therefore, it is easy to get the values of C, B, and M.

2.7 Soft Block Adjustment

For floorplan, the shapes of some blocks may not be fixed. For example, for some blocks, their areas are fixed, but their width/height ratio can be changed in some range. This kind of blocks are called *soft block*. The flexibility of soft block shapes can help us improve BDF solution quality further. Our strategy is as follows.

After applying BDF algorithm, we get a BDF solution. Based on this solution, each time we select a soft block on LCS path which decides the size of the chip [9, 8], reduce the width (height) of the block a little bit, and apply Modified_LCS_Computation to get a new solution. This process is executed repeatedly.

In order to control iterations, simulated annealing is adopted again. The perturbation

operation is to choose a soft block on LCS path and change its width or height accordingly. The cost function is the same as that of BDF algorithm. Figure 2.11 shows an example. Figure 2.11(a) shows a BDF solution. Blocks B, D and E are on an LCS path. E is selected and its width is reduced. Figure 2.11(b) illustrates the BDF solution after repacking. Both the chip area and the bus area are reduced.

2.8 Experimental Results

Our algorithm was implemented in C++ and tested on Intel Xeon (2.4GHz) with 1GB memory. The technique of simulated annealing is used to search for an optimal or near optimal BDF solution with a special annealing schedule. The annealing process starts from an initial temperature. Then the temperature drops linearly while only a small number of moves are made at each temperature. As to the cost function $\alpha \cdot C + \beta \cdot B + \gamma \cdot M$, let $\alpha = \beta = 1$. Since M is the number of unassigned buses, let γ equal to the chip area so that $\gamma \cdot M$ is in the same order of the other two items.

We tested on three sets of test files. The first set is derived from MCNC benchmarks for block placement. We added different numbers of buses to the benchmarks. Once we got a BDF solution, we also applied the soft block adjustment technique to further improve the solution quality. Furthermore, we also made some post processing to move buses apart from each other. The test results are listed in Table 2.1. As an illustration, Figure 2.12 displays the final packing result of ami49-2 after soft block adjustment. The ami49-2 includes 49 blocks and 12 buses. The buses are $\{0, 5, 9, 12, 18\}$, $\{1, 10, 21, 25\}$, $\{2, 28, 33\}$,

Table 2.1 Test set 1.

File	Block	Bus	Results		Soft-Adjust	
			time (s)	dead space	time (s)	dead space
apte	9	5	11	4.11%	12 (+1)	0.72%
xerox	10	6	12	3.88%	13 (+1)	0.95%
hp	11	14	28	5.02%	28 (+0)	0.62%
ami33-1	33	8	61	6.02%	62 (+1)	0.94%
ami33-2	33	18	81	6.10%	86 (+5)	1.27%
ami49-1	49	9	98	5.42%	101 (+3)	0.85%
ami49-2	49	12	278	6.09%	281 (+3)	0.84%
ami49-3	49	15	265	7.40%	268 (+3)	1.09%

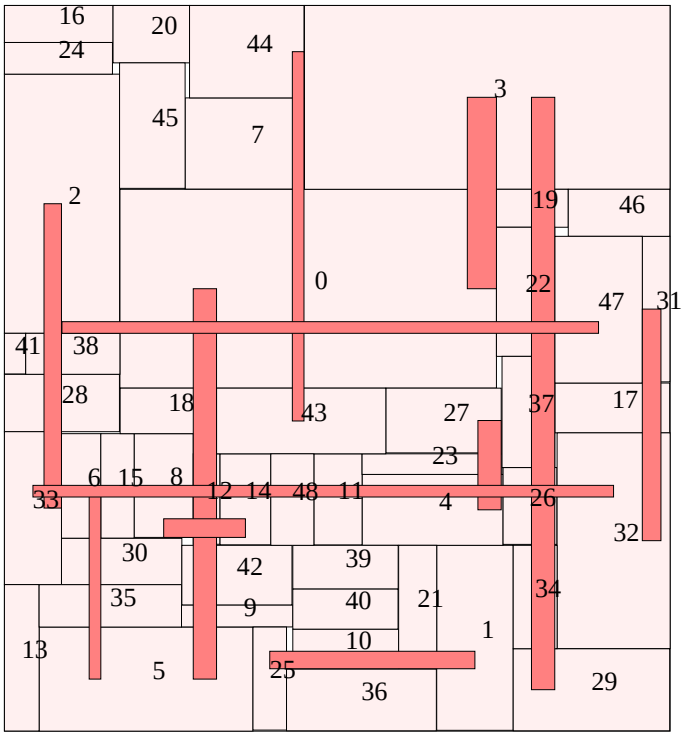


Figure 2.12 The result packing of ami49-2 after soft block adjustment. There are 49 blocks and 12 buses.

Table 2.2 Test sets 2 and 3.

File	Block	Bus	time (s)	deadspace
cad1	40	13	209	4.40%
cad2	57	16	191	5.16%
grid4	16	8	1	0%
grid5	25	10	23	0%
grid6	36	12	103	0%
grid7	49	14	150	0%

$\{3, 19, 22, 26, 29, 34\}$, $\{4, 23, 27\}$, $\{5, 35, 30, 6\}$, $\{32, 31, 17\}$, $\{11, 14, 15, 32, 33\}$, $\{12, 8, 14\}$, $\{44, 43, 7\}$, $\{0, 3\}$, and $\{2, 47\}$. The second test set (cad1 and cad2) includes two test files that are derived from industry designs. To further test our approach, we created a set of bus grid test files. Each test file includes n^2 $(n = 4, ..., 7)$ blocks and $2n$ buses. All blocks have the same block size (600×700) and each bus goes through n blocks. For example, the test file grid7 includes 49 blocks and 14 buses. The buses are $\{0, 1, 2, 3, 4, 5, 6\}$, $\{7, 8, 9, 10, 11, 12, 13\}$, $\{14, 15, 16, 17, 18, 19, 20\}$, $\{21, 22, 23, 24, 25, 26, 27\}$, $\{28, 29, 30, 31, 32, 33, 34\}$, $\{35, 36, 37, 38, 39, 40, 41\}$, $\{42, 43, 44, 45, 46, 47, 48\}$, $\{0, 7, 14, 21, 28, 35, 42\}$, $\{1, 8, 15, 22, 29, 36, 43\}$, $\{2, 9, 16, 23, 30, 37, 44\}$, $\{3, 10, 17, 24, 31, 38, 45\}$, $\{4, 11, 18, 25, 32, 39, 46\}$, $\{5, 12, 19, 26, 33, 40, 47\}$, and $\{6, 13, 20, 27, 34, 41, 48\}$. This kind of problem is quite hard, and the position of one bus heavily affects the assignment of other buses. Still, our approach can find an optimal solution within a short time. Figure 2.13 illustrates an optimal solution to the test file grid7. In this solution, not only the chip area is minimized, but also the total bus area is minimized. Table 2.2 shows the test results of these two test sets.

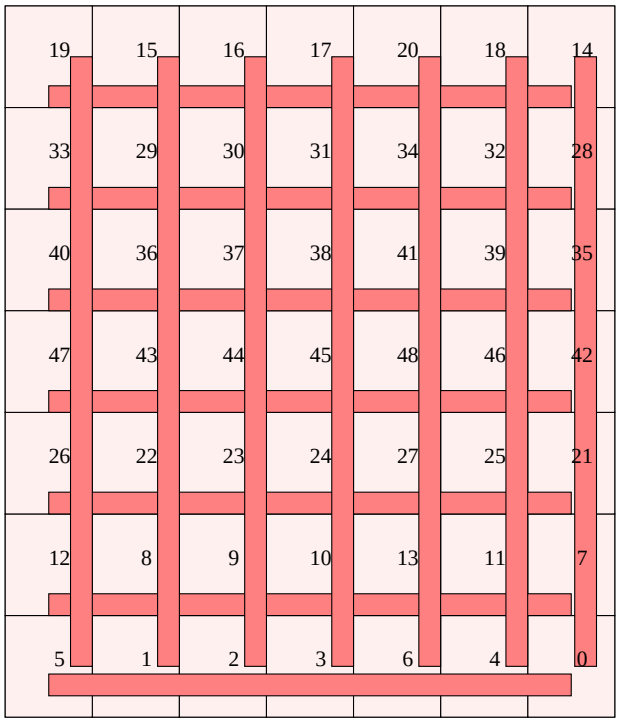


Figure 2.13 An optimal packing of grid7. There are 49 blocks and 14 buses.

2.9 Conclusion

In this paper, we consider the bus-driven floorplanning (BDF) problem. We first derive necessary conditions for feasible buses. Then based on the analysis of the relationship between bus ordering and sequence pair representation, we develop an efficient evaluation algorithm which transforms a sequence pair representation to a BDF solution. Simulated annealing is adopted to search for an optimal or near optimal BDF solution. We also propose a simple but efficient way to handle soft blocks. Experimental results demonstrate that our approach is very efficient and effective.

CHAPTER 3

WIRE PLANNING WITH BOUNDED OVER-THE-BLOCK WIRE CONSTRAINTS

3.1 Introduction

Due to the extreme complexity in System-on-Chip (SoC) design, the hierarchical approach is widely used. By hiding the vast amount of distracting details in low-level objects, design problems can be greatly simplified such that those problems can be solved in efficient ways.

In high-level design, a design is partitioned into several functional blocks, and then each functional block can be designed independently in low levels. However, an implicit constraint is that all these modules have to get connected in a certain way so that they can accomplish the required functionality as a whole. Moreover, as technology scales down, interconnect delay, especially global interconnect delay, has become the dominant factor in achieving high performance in deep submicron design. Therefore, wire planning, which plans the routing of global interconnect among macro blocks, has become an important stage in physical design.

Recently, researchers have been examining extending buffering algorithm to consider

routing [10, 11, 12, 13]. In general it is the right direction to consider the impact of routing on buffering, especially in the final implementation stages. However, explicit buffering consideration in routing at early design stages is very CPU intensive. Furthermore, the final buffering can be done only based on the detailed RC data, which can only be obtained after the low-level block implementation is completed. Before low-level block implementation, it does not make sense to put down all buffers – what we need is to reserve "proper" routing resource for the global routes when doing block implementation.

In our work, we abstract buffering by looking at the problem at a higher level, and consider the problem of wire planning with bounded over-the-block wires (WP). Informally, the problem can be described as follows. Given a placement of macro blocks, some large blocks are evenly divided into several subblocks. A two-pin net is to be routed by going through these blocks/subblocks, i.e., to find a block/subblock sequence as the routing of the net. However, some blocks (or some subblocks) are routing obstacles while some blocks, such as IPs (Intellectual Property) whose internal structures have been fixed, only allow routing but no buffers can be inserted. We call the latter kind of blocks "routing-block". Since a routing-block cannot hold any buffers, the longest over-the-block wires in it have to be limited within a certain range since if the distance between two buffers is large, the signal slew rate is slow which in turn may cause signal integrity problems. In WP problem, each routing-block has an interconnect bound such that the longest over-the-block wires in the routing block can only cross a certain number of subblocks. The bound is in line with designers' practice in keeping long routes "buffer-able" to meet timing and transition time

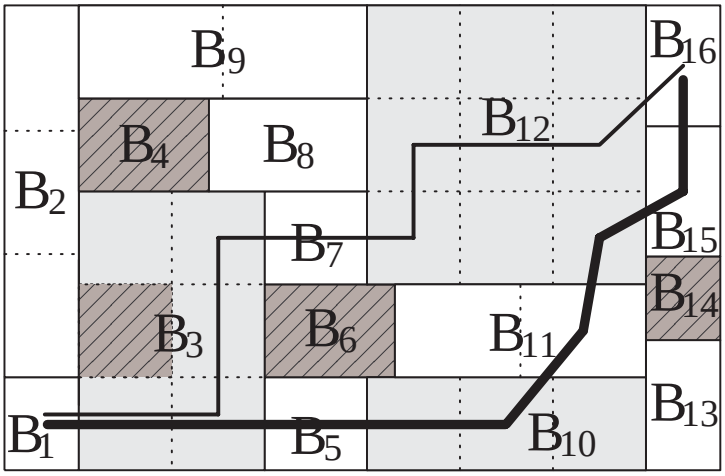


Figure 3.1 The routing illustrated by thin lines is not valid; while the routing shown by wide lines is a feasible solution.

constraints. Different routing-blocks may set different bounds.

Figure 3.1 illustrates an example of 16 macro blocks. Dark blocks are obstacles while gray ones can be used for routing but cannot hold buffers. Since some macro blocks are big, they are partitioned into several small subblocks in order to get more accurate estimation. The blocks B_2, B_3, B_9, B_{10}, B_{11}, and B_{12} are divided into 1×3, 2×3, 2×1, 3×1, 2×1, and 3×3 subblocks respectively. B_4, B_6, and B_{14} as well as a subblock of B_3 are obstacles. B_3, B_{10}, and B_{12} are routing-blocks, and the longest over-the-block wires in these three blocks should not go across more than two subblocks. One two-pin net is to be routed between B_1 and B_{16}. However, the routing illustrated by thin lines does not satisfy the interconnect constraints since the over-the-block wires in both blocks B_3 and B_{12} cross 4 subblocks. On the other hand, the routing illustrated by wide lines is a valid solution. For short, let OB-wire refer to over-the-block wire.

In this chapter, we present two exact polynomial-time algorithms for wire planning with

bounded over-the-block wires. Both algorithms guarantee to find a feasible routing solution with minimum wire length as long as a solution exists. Furthermore, both algorithms use shortest path algorithm, but the constructions of the underlying graphs, which incorporate interconnect constraints, are different. One approach requires less memory, but the other approach may take less time to adjust the graph after routing one net. According to different requirements, users can choose an appropriate algorithm to solve the problem.

The rest of the chapter is organized as follows. Section 3.2 defines the WP problem. In Section 3.3, we present two exact polynomial-time algorithms based on the shortest path algorithm with different graph constructions. Finally we show the experimental results in Section 3.4 and conclude the chapter in Section 3.5.

3.2 Problem Formulation

A placement of m macro blocks $\mathcal{B} = \{B_1, ..., B_m\}$ is given. Since some blocks are large, they are divided into several subblocks. Suppose the block B_i is evenly divided into $p_i \times q_i$ blocks, and these subblocks form a $p_i \times q_i$ block grid. Let $R_i = p_i \times q_i$ be the number of subblocks of B_i, and r_i^j ($j = 1, ..., R_i$) refer to a subblock of block B_i. For convenience, if B_i is not divided into subblocks, we still say it is divided into $p_i \times q_i$ subblocks where $p_i = q_i = 1$.

Among these m blocks, some blocks are routing-blocks which allow over-the-block routing but no buffers can be inserted. An interconnect bound d_i is set for each routing-block B_i such that all over-the-block wires within B_i go through at most d_i subblocks.

Furthermore, some blocks do not even allow over-the-block routing and they become routing obstacles. Some subblocks within a block can also be obstacles. Finally if a block is neither a routing block, nor an obstacle, then buffers can be inserted and OB-wires in the block are not constrained.

With all these requirements, the target is to find the shortest over-the-block routing for a two-pin net such that the routing satisfies the interconnect constraints. The routing wire length is measured by the Manhattan distance between the centers of two macro blocks.

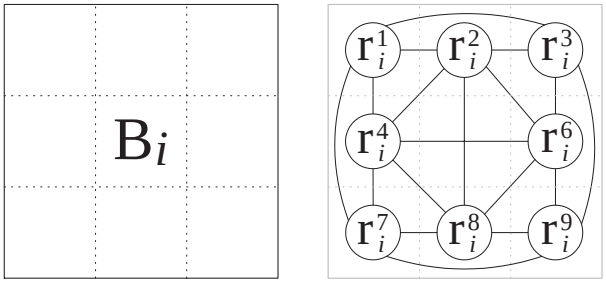
3.3 WP Algorithm

If no interconnect constraints are considered, the routing of a two-pin net can be interpreted as a shortest path between two blocks. But when interconnect constraints are involved, they prevent us from applying shortest path algorithm directly. Therefore, our approaches are to construct a graph so that interconnect constraints can be incorporated in the graph. Then the shortest path algorithm is called to find the optimal routing solution. The main scheme of a WP algorithm is shown as *Algorithm* 3:

Algorithm 3 WP-Algorithm ( )

1: Construct a graph incorporating constraints;
2: Apply shortest path algorithm on the graph;
3: Derive routing solution for the given net;

In the following two subsections, we present two ways of constructing the graph for a given WP problem.



(a) (b)

Figure 3.2 (a) A routing-block B_i is divided into 3×3 subblocks, and the interconnect bound is 3. (b) A graph denotes all valid OB-wires within B_i.

3.3.1 WP-Path algorithm

As we notice that all interconnect constraints are confined to OB-wires within routing-blocks. Therefore, the main idea of WP-Path algorithm is to explore all legal connections between any two boundary subblocks within the same routing-block.

Figure 3.2 illustrates an example. Figure 3.2(a) shows a routing-block B_i that is divided into 3×3 subblocks. Suppose the interconnect bound is 3. In Figure 3.2(b), each subblock is represented by a node and the edges imply that the corresponding two blocks can be connected by an OB-wire which crosses at most three subblocks. In other words, any valid OB-wire in B_i corresponds to an edge in Figure 3.2(b).

However, if two or more successive edges are selected for one OB-wire, the interconnect constraint may not be satisfied. For example, in Figure 3.2(b), if the edges (r_i^1, r_i^3) and (r_i^3, r_i^9) are selected, then the corresponding OB-wire goes through five subblocks (i.e., r_i^1, r_i^2, r_i^3, r_i^6, and r_i^9). In order to avoid selecting successive edges related to a routing-block, one node is split into two nodes which are called in-node and out-node, respectively. Figure 3.3 illustrates an example. Two nodes r_i^j and r_i^k belong to the same routing-block B_i, and

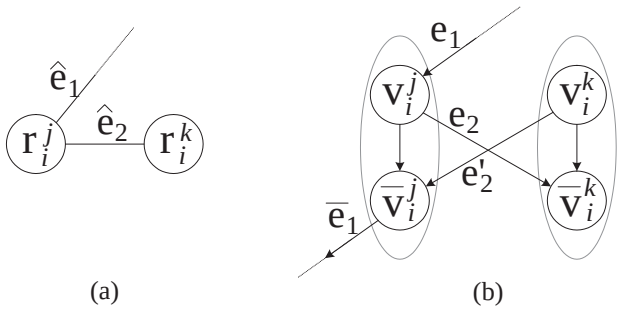


Figure 3.3 (a) The two nodes r_i^j and r_i^k are within the same routing-block B_i. (b) Each node is split into an in-node and an out-node.

there is an edge $\hat{e}_2$ connecting the two nodes. But the edge $\hat{e}_1$ connects r_i^j to a node which does not belong to the block B_i. Then the node r_i^j is split to an in-node v_i^j and an out-node $\bar{v}_i^j$. Similarly for the node r_i^k. Also the undirected edge $\hat{e}_2$ becomes two directed edges e_2 and e_2' pointing from an in-node to an out-node. Furthermore, since the edge $\hat{e}_1$ connects a node not belonging to B_i, it becomes two edges e_1 and e_1'. e_1 is incident to the in-node v_i^j, while e_1' is incident from $\bar{v}_i^j$.

Based on the above strategy, we construct a directed graph *Path Graph* $G_p = (V_p, E_p)$ for a WP problem as follows:

1. Nodes $V_p = V_{ps} \cup V_{in} \cup V_{out}$ where

   $V_{sp} = \{u_i^j | i = 1, ..., m; r_i^j$ is a subblock of B_i, and B_i is neither an obstacle nor a routing-block $\}$

   $V_{in} = \{v_i^j | i = 1, ..., m; r_i^j$ is not an obstacle and it is a subblock on the boundary of B_i $\}$

   $V_{out} = \{\bar{v}_i^j | i = 1, ..., m; r_i^j$ is not an obstacle and it is a subblock on the boundary of B_i $\}$

For convenience, node $v_i^j \in V_{in}$ is called in-node, while node $\bar{v}_i^j \in V_{out}$ is called out-node. Without misunderstanding, the corresponding subblock of a node v_i^j (or $\bar{v}_i^j$ or u_i^j) is r_i^j.

2. Edges $E_p = E_{pa} \cup E_{pb} \cup E_{pc} \cup E_{pd} \cup E_{pe}$ where

   $E_{pa} = \{(u_i^j, v_k^l) | u_i^j \in V_{ps}, v_k^l \in V_{in}, r_i^j$ and r_k^l are adjacent $\}$

   $E_{pb} = \{(\bar{v}_i^j, u_k^l) | \bar{v}_i^j \in V_{out}, u_k^l \in V_{ps}, r_i^j$ and r_k^j are adjacent $\}$

   $E_{pc} = \{(u_i^j, u_k^l) | u_i^j, u_k^l \in V_{ps}, u_i^j \neq u_k^l, r_i^j$ and r_k^l are adjacent $\}$

   $E_{pd} = \{(v_i^j, \bar{v}_i^j) | v_i^j \in V_{in}, \bar{v}_i^j \in V_{out}\}$

   $E_{pe} = \{(v_i^j, \bar{v}_i^k) | v_i^j \in V_{in}, \bar{v}_i^k \in V_{out}, j \neq k,$ and there exists routing between two subblocks r_i^j and r_i^k such that the routing goes across at most d_i subblocks $\}$

   Edges in $E_{pd} \cup E_{pe}$ connects nodes whose corresponding subblocks belong to the same routing-blocks. As we notice that those edges are always incident from an in-node to an out-node. Therefore, no path can take two successive edges related to the same routing-block. Furthermore, any valid OB-wire in a routing-block corresponds to an edge in $E_{pd} \cup E_{pe}$. On the other hand, one edge in $E_{pd} \cup E_{pe}$ implies that at least one valid OB-wire exists between the two corresponding subblocks. For convenience, we say an edge of $E_{pd} \cup E_{pe}$ is within the related routing-blocks.

3. Edge cost

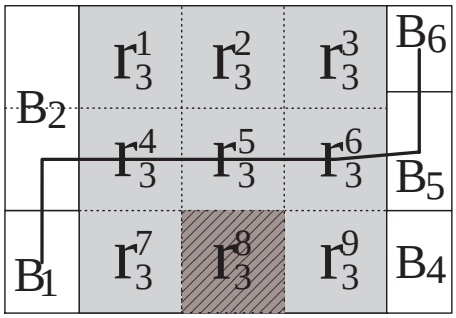


Figure 3.4 The solid lines indicate a routing solution between blocks B_1 and B_6.

If $e \in E_{pd}$, $C(e) = 0$

If $e \in E_{pa} \cup E_{pb} \cup E_{pc}$, and $e = (\alpha_i^j, \beta_k^l)$, $C(e) = |x_i^j - x_k^l| + |y_i^j - y_k^l|$ where (x_i^j, y_i^j) and (x_k^l, y_k^l) are the center coordinates of subblocks r_i^j and r_k^l

If $e \in E_{pe}$, and $e = (v_i^j, \bar{v}_i^k)$, $C(e)$ is the shortest routing length within routing-block B_i between two subblocks r_i^j and r_i^k

Figure 3.4 illustrates an example. Figure 3.4 includes six macro blocks. B_3 is a routing-block, and it is divided into 3×3 subblocks. Subblock r_3^8 of B_3 is obstacle. Also the block B_2 is divided into 1×2 subblocks. Suppose the interconnect bound for B_3 is 3 and we want to find the routing between B_1 and B_6 under the interconnect constraint. Figure 3.5 is the corresponding Path Graph G_p. Since r_3^8 is obstacle and r_3^5 is not a boundary subblock, they are not included in G_p.

In Figure 3.5, edges within ellipses belong to E_{pd}. The two nodes in one ellipse are the in-node and out-node corresponding to a subblock. Since this kind of edge is used to connect in-nodes and out-nodes, the cost is zero. For other edges within the routing-block B_3 (i.e., edges belonging to E_{pe}), an edge $(v_3^i, \bar{v}_3^j)$ $(i, j = 1, ..., 9; v_3^i \in V_{in}; \bar{v}_3^j \in V_{out}; i \neq j)$

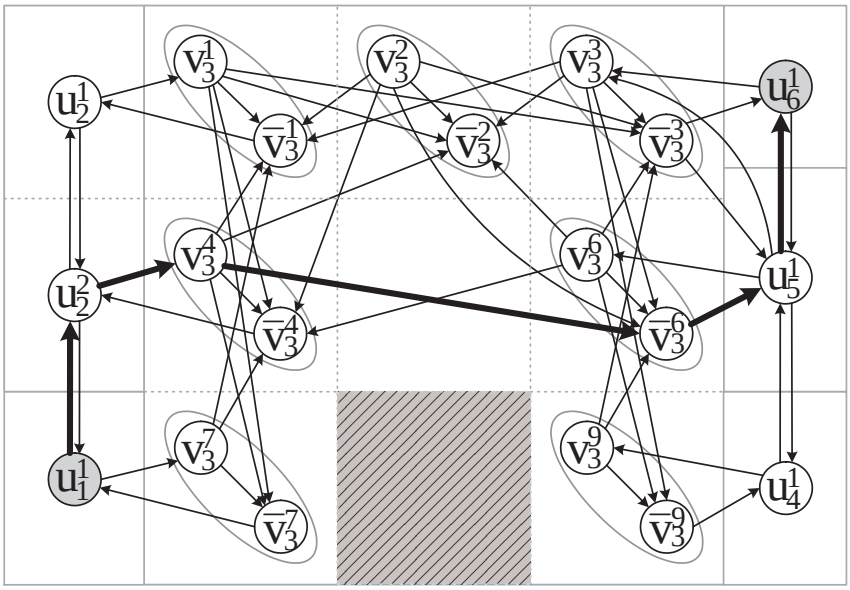


Figure 3.5 The corresponding path graph G_p. The wide lines illustrate a shortest path from u_1^1 to u_6^1.

is added if the subblock r_3^i can reach r_3^j by crossing at most three subblocks of B_3. For example, r_3^7 can reach r_3^1 by going through r_3^4. So one edge $(v_3^7, \bar{v}_3^1)$ is added. But the shortest routing between r_3^7 and r_3^9 has to go through r_3^4, r_3^5 and r_3^6. Therefore $(v_3^7, \bar{v}_3^9)$ is not inserted in G_p. The cost for this kind of edges is the shortest routing length between the two subblocks.

Once the path graph G_p is constructed, we can apply the shortest path algorithm [14, 15] to decide the routing of a net. In this example, we plan to route one net between B_1 and B_6. Therefore, we let u_1^1 be the starting point and u_6^1 be the ending point. The shortest path from u_1^1 to u_6^1 is illustrated by wide lines in Figure 3.5. And it is easy to derive the over-the-block routing from this solution as shown in Figure 3.4.

In order to construct a path graph G_p, we need to find all valid connections within each

routing-block, i.e., how to create edges in E_{pe}. For each routing-block B_i, an undirected graph $\tilde{G}_i$ is set up by presenting each subblock, which is not obstacle, as a node and adding edges according to block adjacency. The cost of all edges is 1. Then apply all-pair-shortest path algorithm on $\tilde{G}_i$. If the distance between two boundary subblocks is less than or equal to the interconnect bound of B_i, then add edges in G_p accordingly.

Suppose each block B_i is divided into R_i ($R_i = p_i \times q_i$) subblocks. If B_i is a routing-block, the nodes related to B_i are $O(p_i + q_i)$; otherwise, the number of nodes is (R_i). Therefore, we get

$$|V_p| = O(\sum_{i=1}^{m} R_i)$$

For edges in G_p, $|E_{pa} \cup E_{pb} \cup E_{pc}| = O(|V_p|)$ since each edge connects two adjacent blocks/subblocks, and the adjacency relationship among blocks/subblocks on a plane constitutes a planar graph. Also $|E_{pd}| = |V_{in}| = |V_{out}|$. Finally $|E_{pe}| = O(\sum_{i=1}^{m} \min\{(p_i + q_i)^2, d_i \cdot (p_i + q_i)\})$ where d_i is the interconnect bound for block B_i. Without loss of generality, we assume $d_i = O(p_i + q_i)$. Then $|E_{pe}| = O(\sum_{i=1}^{m}(p_i + q_i)^2)$. Therefore, we get

$$|E_p| = O(\sum_{i=1}^{m}(p_i + q_i)^2)$$

However, during the construction of path graph G_p, we first need to set up $\tilde{G}_{pi} = (\tilde{V}_{pi}, \tilde{E}_{pi})$ for each routing-block B_i in order to decide edges in E_{pe}. Since $|\tilde{V}_{pi}| = O(R_i)$, $|\tilde{E}_{pi}| = O(R_i)$, and all-pair shortest path algorithm takes $O(|\tilde{V}_{pi}||\tilde{E}_{pi}| + |\tilde{V}_{pi}|^2 \log |\tilde{V}_{pi}|)$ running time [14, 15], the creation of edges inside routing-block B_i takes $O(R_i^2 \log R_i)$ running time. Therefore, the total construction time for G_p is $O(\sum_{i=1}^{m} R_i^2 \log R_i)$.

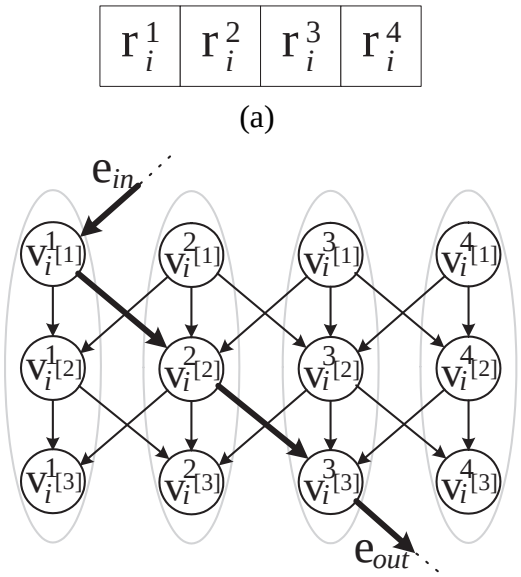


Figure 3.6 (a) r_i^1, r_i^2, r_i^3, and r_i^4 are subblocks of a routing-block B_i. (b) Each subblock is represented by a node array.

Once the path graph $G_p = (V_p, E_p)$ is constructed, the single-source shortest path algorithm can be used to find the routing with over-the-block wiring constraint for a two-pin net, and it can be accomplished in $O(|V_p||E_p|)$ time.

3.3.2 WP-Split algorithm

In this section, we propose another algorithm by constructing a directed graph *Split Graph* G_s. The main idea of this approach is to represent a subblock in a routing-block, which is not an obstacle, by d_i nodes. Then each path segment in the routing-block starts with a node whose index is 1, and ends at a node whose index is d_i. Since all nodes with index d_i only have out-going edges pointing to a node which is not related to B_i, the length of a path segment is forced to be d_i. Figure 3.6 illustrates an example. Suppose the interconnect

bound for B_i is 3. r_i^1, r_i^2, r_i^3 and r_i^4 are subblocks of a routing-block B_i. Each subblock is represented by a node array. Then a path segment in B_i must go from a node with index 1 to a node with index 3.

The construction of a Split Graph $G_s = (V_s, E_s)$ is as follows. If B_i is a routing-block, let $L_i = d_i$ where d_i is the interconnect bound of B_i. Otherwise, let $L_i = 1$.

1. Nodes $V_s = \cup_{i=1}^{m} V_{si}$ where

   $V_{si} = \{v_i^j[k] | r_i^j$ is a subblock of B_i and it is not an obstacle; $k = 1, ..., L_i\}$

2. Edges $E_s = E_{sa} \cup E_{sb} \cup E_{sc}$ where

   $E_{sa} = \{(v_i^j[k], v_i^j[k+1]) | v_i^j[k], v_i^j[k+1] \in V_{si}, k = 1, ..., L_i - 1\}$

   $E_{sb} = \{(v_i^j[k], v_i^l[k+1]) | v_i^j[k], v_i^l[k+1] \in V_{si}, k = 1, ..., L_i - 1,$

   $j \neq l$, and subblocks r_i^j and r_i^l are adjacent.$\}$

   $E_{sc} = \{(v_i^j[L_i], v_k^l[1]) | v_i^j[L_i] \in V_{si}, v_k^l[1] \in V_{sk}$, and two subblocks r_i^j and r_k^l

   are adjacent, but do not belong to the same routing-block $\}$

   As we notice that edges belonging to E_{sb} always connect nodes whose indexes increase by one. Then a path segment within a routing-block B_i is always from a node with index 1 to a node with index L_i. In other words, any OB-wire in B_i cannot exceed the interconnect bound d_i.

3. Edge Cost

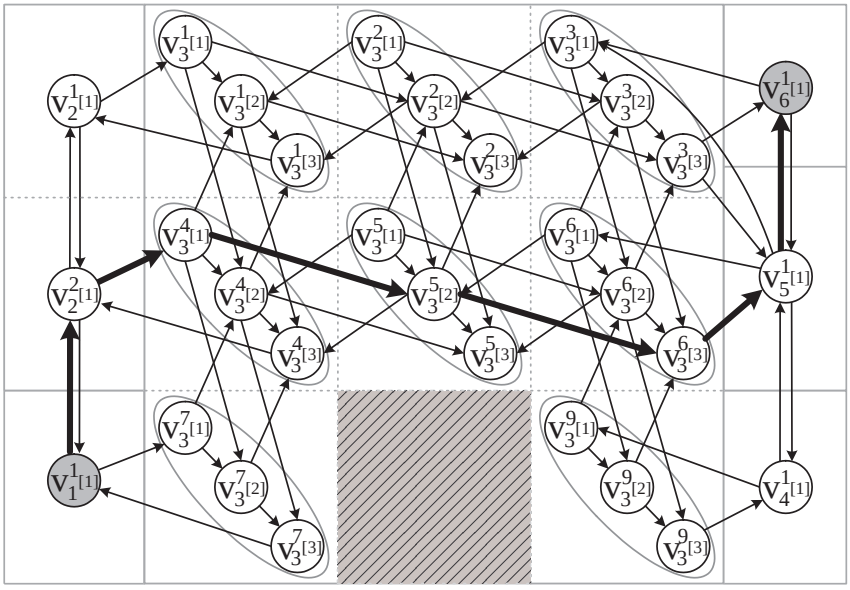


Figure 3.7 The corresponding split graph for Figure 3.4. The wide lines illustrate a shortest path from $v_1^1[1]$ to $v_6^1[1]$.

If $e \in E_{sa}$, $C(e) = 0$

If $e \in E_{sb}$, $e = (v_i^j[k], v_i^l[k+1])$, $C(e) = |x_i^j - x_i^l| + |y_i^j - y_i^l|$ where

(x_i^j, y_i^j) and (x_i^l, y_i^l) are the center coordinates of subblocks r_i^j and r_i^l

If $e \in E_{sc}$, $e = (v_i^j[L_i], v_k^l[1])$, $C(e) = |x_i^j - x_k^l| + |y_i^j - y_k^l|$ where

(x_i^j, y_i^j) and (x_k^l, y_k^l) are the center coordinates of subblocks r_i^j and r_k^l

We still use the example of Figure 3.4 to illustrates our approach. Since the interconnect bound for B_3 is 3, three nodes are created for each subblock excluding obstacles. Then edges are added among these nodes. Edges in ellipses belong to E_{sa} and their cost is 0. For any other edge e inside B_3 (i.e., $e \in E_{sb}$), the index of its source node is less than that of its target node, and the difference is 1. On the other hand, for subblocks of B_2, they are represented by only one node since B_2 is not a routing-block and has no

interconnect constraint. Finally, if two subblocks are adjacent and they do not belong to the same routing-block, then the tail of a node array points the head of another node array. This kind of edges belongs to E_{sc}. The wide lines in Figure 3.7 shows a shortest path from $v_1^1[1]$ to $v_6^1[1]$, and it corresponds to the routing in Figure 3.4.

Since a routing-block is divided into $R_i = p_i \times q_i$ subblocks and each subblock r_i^j is represented by d_i nodes, $|V_s| = O(\sum_{i=1}^{m} d_i \times R_i)$. $|E_{sa}| = O(|V_s|)$. $|E_{sb} = O(|V_s|)$ and $|E_{sc}| = O(|V_s|)$. Therefore $|E_s| = O(|V_s|)$.

Once the Split Graph $G_s = (V_s, E_s)$ is constructed, the single-source shortest path algorithm can be used to find the over-the-block routing for a two-pin net, and it can be accomplished in $O(|V_s||E_s|) = O((\sum_{i=1}^{m} d_i \times R_i)^2)$ time.

3.3.3 Comparison

In the above, we have presented two approaches to set up a graph which implies interconnect constraints. The comparison between the two graphs are listed in Table 3.1. $R_i = p_i \times q_i$ is the number of subblocks in a block, and d_i is the interconnect bound of a routing-block B_i. Without loss of generality, we assume $d_i = O(p_i + q_i)$.

The advantage of WP-Path algorithm is that it requires less memory since the underlying graph is smaller. However, it takes longer time to construct the graph. "Setup" in the table shows the construction time. "Path" is the time to search for a shortest path based on the graph. And since the size of the underlying graph of WP-Path is smaller than that of WP-Split, it takes shorter time to find a routing solution for one net.

Table 3.1 Algorithm comparison.

Algorithm	WP-Path	WP-Split
Nodes	$O(\sum_{i=1}^{m} R_i)$	$O(\sum_{i=1}^{m} d_i \times R_i)$
Edges	$O(\sum_{i=1}^{m} (p_i + q_i)^2)$	$O(\sum_{i=1}^{m} d_i \times R_i)$
Setup	$O(\sum_{i=1}^{m} R_i^2 \log R_i)$	$O(\sum_{i=1}^{m} d_i \times R_i)$
Path	$O((\sum_{i=1}^{m} R_i)(\sum_{i=1}^{m} (p_i + q_i)^2))$	$O((\sum_{i=1}^{m} d_i \times R_i)^2)$
Adjust	$O(R_i^2 \log R_i)$	$O(d_i^2)$

Furthermore, after routing one net, some subblocks may become routing obstacles since the net consumes all routing resource. Therefore, we have to judge whether the edges within a routing-block are still valid. For example, in Figure 3.4, the routing of the net goes through routing-block B_3. Suppose the subblock r_3^4 can accommodate only one net. After routing this net, r_3^4 becomes a routing obstacle. Then in Figure 3.5, serval edges such as $(v_3^1, \bar{v}_3^7)$ and $(v_3^2, \bar{v}_3^4)$ inside B_3 are not valid any more. For the two approaches, WP-Path have to recalculate edges within every routing-block along the path, while WP-Split only needs to remove some edges from the graph. The time required for adjusting edges within one routing-block is shown in Table 3.1 "Adjust". In general, WP-Split is faster for graph adjustment, and it is suitable for handling a large number of nets.

3.4 Experimental Results

Our algorithms were implemented in C++ on PC (733MHz) with 128MB memory. We tested WP-Path and WP-Split algorithms on three test files which were generated randomly. We compared the results of our two approaches with another shortest path approach that

Table 3.2 Test results of WP-Path and WP-Split algorithms.

File		test1	test2	test3
Macro Blocks		50	100	150
Routing-Blocks		7	12	18
Nets		350	1600	3500
Total Subblocks		1337	1736	2604
Obstacles		18	48	131
Shortest Path	Time(s) (per net)	11 (0.031)	23 (0.014)	76 (0.022)
	Nets	165 (47.14%)	613 (38.31%)	1211 (34.6%)
WP-Path	Time(s) (per net)	9 (0.026)	104 (0.065)	202 (0.058)
	Nets	350 (100%)	1600 (100%)	3500 (100%)
WP-Split	Time(s) (per net)	18 (0.051)	91 (0.057)	227 (0.065)
	Nets	350 (100%)	1600 (100%)	3500 (100%)

finds a shortest path from the source node to the end node without considering interconnect constraints. If the interconnect constraint is not satisfied, the net is discarded. Each time, one net is selected to be routed. Furthermore, we assume that all subblocks can accommodate a large number of nets so that the routing of one nets is not affected by others. In this way, we can concentrate on the routing quality of WP algorithms. Both WP-Path and WP-Split algorithms guarantee to find feasible routing as long as one solution exists. As shown in Table 3.2, all nets can be routed by our approaches, while less than 50% of the nets can find feasible routing by the simple shortest path approach.

The underlying graph of WP-Path usually has smaller size than that of WP-Split. For the test file “test2”, the number of nodes in WP-Path is about 1200 and edges are about 25000; while WP-Split needs about 16500 nodes and 73000 edges. WP-Path gains in mem-

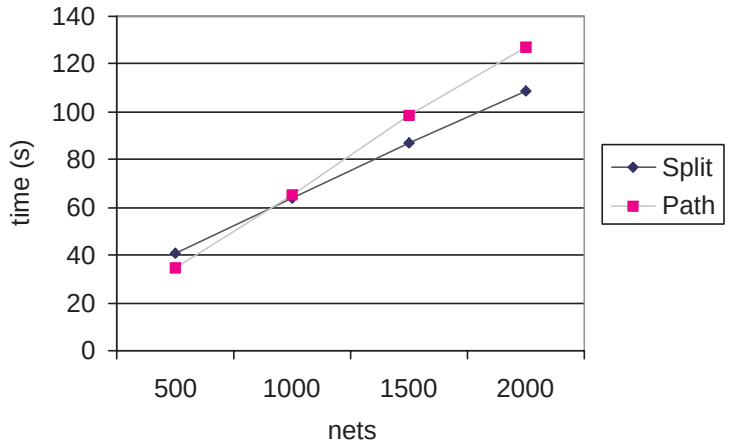


Figure 3.8 The comparison of WP-Path and WP-Split on the relationship between running time and the number of nets.

ory usage. However, WP-Split has the advantage that it can make changes to the underlying graph more easily after routing one net. When a larg number of nets are to be processed, WP-Split may requires less running time than that of WP-Path. Figure 3.8 shows the relationship between the number of nets and the running time for “test2”. As the number of nets increases, the running time of WP-Split becomes shorter than that of WP-Path.

3.5 Conclusion

In this chapter, we presented two exact polynomial-time algorithms for wire planning with bounded over-the-block wires. Both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists. One requires less memory, while the other may take less running time when processing a large number of nets. According to different application requirements, users can choose an appropriate one.

CHAPTER 4

SIMULTANEOUS PIN ASSIGNMENT AND ROUTING

4.1 Introduction

Due to the enormous complexity of VLSI design, a hierarchical approach is needed for the placement and routing of millions of standard cells in order to reduce runtime and improve solution quality. Pin assignment and routing for macro blocks are important steps in a typical top-down hierarchical approach.

Existing algorithms for macro-block pin assignment and routing can be classified into

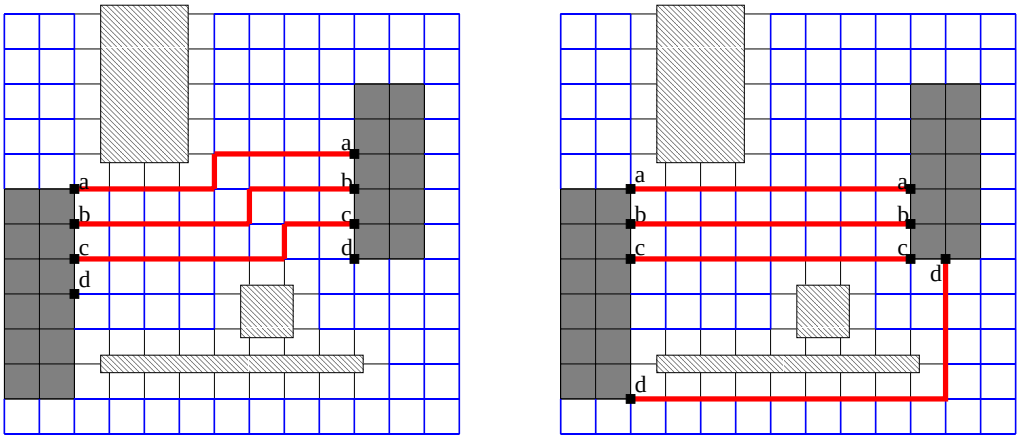
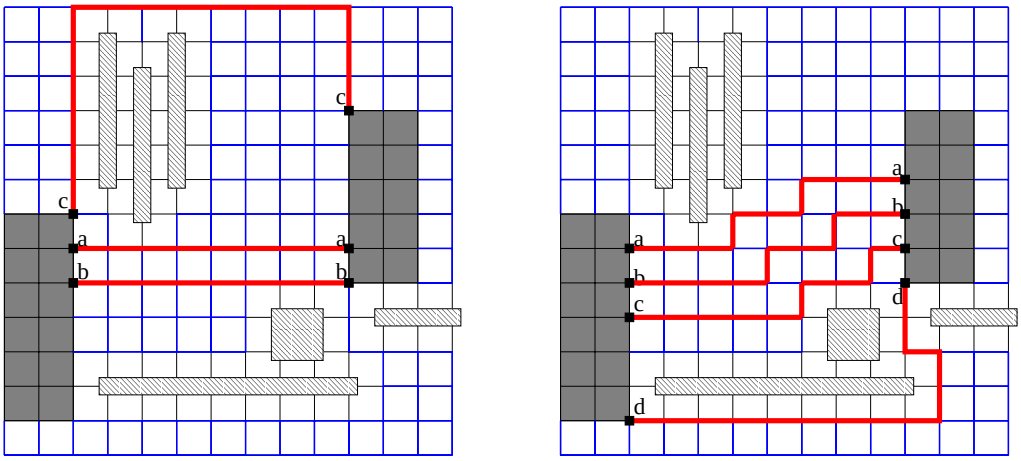


Figure 4.1 (a) The two-step approach fails to route all nets. (b) The optimal solution of pin assignment and routing by our approach.

two categories: (1) a two-step approach where pin assignment is followed by routing [16, 17, 18, 19], and (2) a net-by-net approach where pin assignment and routing for a single net are performed simultaneously [20, 21, 22, 23, 24]. None of the existing algorithms is "exact" in the sense that the algorithm may fail to route all nets even though a feasible solution exists. This remains true even if only two-pin nets with fixed pins between two blocks are concerned. Let us use two examples to illustrate that previous approaches cannot guarantee a feasible solution. The first example in Figure 4.1 includes two macro blocks and three obstacles in a one-layer routing environment. Four nets $\{ a, b, c, d \}$ are to be assigned pins and routed. Figure 4.1(a) is obtained by a two-step approach. But at most three nets can be routed for the pin assignment solution; i.e., the pin assignment solution leads to a routing problem that is not routable by any router. Figure 4.1(b) shows a feasible solution. Figure 4.2 gives another example that includes two macro blocks and six obstacles in one-layer routing environment. Four nets $\{a, b, c, d\}$ are to be assigned pins and routed. Figure 4.2(a) is the result of the net-by-net approach. Still it is impossible to assign pins and route for net d. Figure 4.2(b) gives a feasible solution. Note that both feasible solutions can be obtained by our algorithm in this chapter.

In this chapter, we first consider the two-pin net connections from one block to all other blocks. We present the first polynomial-time exact algorithm for simultaneous pin assignment and routing for all two-pin nets between one block (source block) and all other blocks. In addition to finding a feasible solution whenever one exists, it guarantees to find a pin-assignment/routing solution with minimum cost $\alpha \cdot W + \beta \cdot V$ for any positive pair



(a) (b)

Figure 4.2 (a) The net-by-net approach fails to route all nets. (b) The optimal solution of pin assignment and routing by our approach.

α and β, where W is the total wire length and V is the total number of vias. Our algorithm has various applications. (1) It is suitable in ECO situations where the existing solution is modified incrementally. (2) Given any pin assignment and routing solution obtained by any existing method, we can increase the number of routed nets and reduce the routing cost by removing the routes connecting to one block and redoing pin assignment and routing with our algorithm. Furthermore, applying the algorithm iteratively (each time randomly pick one block as source block) provides a polynomial-time randomized algorithm for the pin assignment and routing problem among blocks. This method is applicable to both global and detailed routing with arbitrary routing obstacles on multiple layers. Experimental results demonstrate its efficiency and effectiveness.

Our method is based on min-cost flow computations. Although network flow formulations have been proposed for routing in the past [25, 26, 27, 28, 29, 30], there are important differences between our work and previous results. First, previous network flow formula-

tions were primarily designed for global routing, whereas ours combines pin assignment with routing (detailed or global). Second, almost all of those previous works needed to solve the multicommodity flow problem which is NP-hard, whereas our algorithm uses min-cost flow which is a polynomial time solvable problem. Meixner and Lauther proposed an algorithm using min-cost flow. However, the nets handled by the algorithm have to be connected to one common node. Third, our algorithm exactly solves the simultaneous pin assignment and routing problem for all two-pin nets from one block to all other blocks in polynomial time. Note that the routing step alone is NP-complete even if there are only two blocks and all nets are two-pin nets with fixed pins.

The rest of the chapter is organized as follows. Section 4.2 defines the problem of simultaneous pin assignment and routing in multilayer. In Section 4.3, we give a network flow formulation to find the routes for all two-pin nets between one block and all other blocks. In Section 4.4, we discuss its application in ECO situation, and demonstrate how to use it to improve any given solution and to solve the pin assignment and routing problem among blocks. Furthermore, we extend the algorithm to handle multiple pin nets. Finally, we show the experimental results in Section 4.5 and conclude the chapter in Section 4.6.

4.2 Problem Definition

The macro block layout in multilayer is modelled by a three-dimensional multilayer routing grid graph $G = (V, E)$. For convenience, we call the 3 dimensions as x, y, and z dimension. Each layer is an x-y dimensional grid. Along z axis, the grid nodes with the same x, y

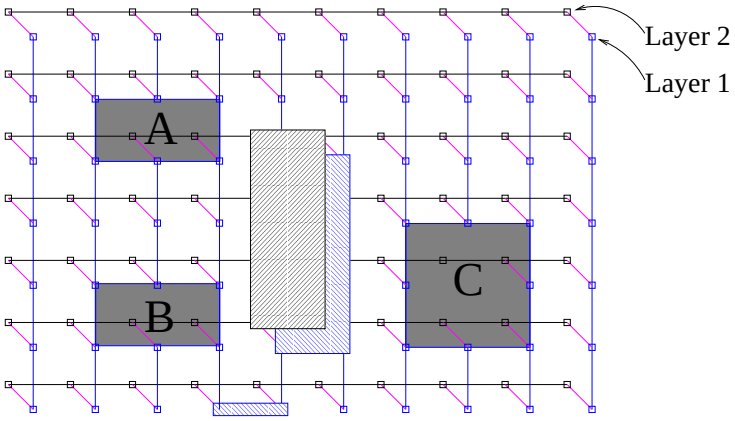


Figure 4.3 A routing grid graph for two layers.

coordinates on different layers are connected by *via* edges. The adjacent nodes on the same layer are connected by edges which represent wire segments. In some technology, a layer has a specified track orientation. In this case, if a layer is used for horizontal tracks, horizontally adjacent nodes of the layer are connected by edges. The similar rule applies for vertical tracks. For routing obstacles (routing congestion region, prerouted wires, and so on) where wiring is not allowed, there are no edges (then the nodes inside the obstacles can be omitted). In practice, routing obstacles can present in any layer, and a block can occupy any number of layers. There is no node inside the region that a block occupies, but the nodes on the boundary, which are possible pin locations, are connected to the nodes outside the block. Furthermore, the nodes over a block in the graph are for over-the-block routing. As an example, Figure 4.3 illustrates a two-layer routing grid graph. Layer 1 is used for vertical tracks, and Layer 2 for horizontal tracks. Three shaded regions (A, B, C) are macro blocks. The other three regions represent routing obstacles. Blocks occupy Layer 1. There are two obstacles in Layer 1 and one in Layer 2.

The grid graph contains not only the topological information (layer and via information), but also the routing obstacle information. So this model is quite flexible and accurate for multilayer technologies, and is suitable for both global routing and detailed routing.

In this model, each edge and each node have a capacity which specifies how many wires are allowed to go through. In detailed routing, the capacity is 1. Also each edge is associated with a cost. The cost of a via edge is β which is specified by users. For wire, the cost is $\alpha \cdot l_e$ where α is specified by users and l_e is the wire length. Since different layers may have different width and resistance, the cost of edges of different layers could be different accordingly. For example, for a two-layer routing grid, we assign α_1 to Layer 1 and α_2 to Layer 2. Then the cost function becomes

$$\alpha_1 \cdot W_1 + \alpha_2 \cdot W_2 + \beta \cdot V$$

where W_1 is the total wire length on Layer 1, W_2 is the total wire length on Layer 2, and V is the total number of vias. The algorithm guarantees to find a solution with minimum total cost.

The goal of pin assignment is to decide the exact pin positions on macro blocks. Routing is to find an appropriate connection among the pins of the same net. The two tasks are closely related. Pin assignment alone neglects many important factors since interconnect is hard to estimate accurately without carrying out the actual routing step. Moreover, a global view of net and routing resource information is critical for pin assignment and routing. In this chapter, we consider the problem of simultaneous pin assignment and routing for all two-pin nets between a block (source block) and all other blocks. The general problem of

pin assignment and routing among blocks will be discussed in Section 4.4.

The problem (called *PAR*: Pin Assignment and Routing) can be formally described as follows:

Problem 4.1 PAR: *Given a routing grid graph $G = (V, E)$ with U and C where U is a function on edges and nodes denoting the capacity of edges and nodes and C is a function on edges denoting the cost of edges, a set of $m + 1$ macro blocks B(one block is the source block b_s, and the others are sink blocks $b_1, b_2, ..., b_m$), and a set of nets $N = N_1 \cup N_2 \cup ... \cup N_m$ where $N_i, i = 1, 2, ..., m$ is the set of nets between block b_s and blocks b_i, find a set of paths connecting b_s and $b_1, b_2, ..., b_m$, each path corresponding to a net in N, such that each edge/node is used no more than its capacity and the total cost for all nets is minimized. Each endpoint of a path is a pin location.*

In PAR problem, the connections from the points inside the block to the boundary points of the same block are redundant since we can pick the boundary points as pins with less wiring cost. It is true that for multilayer layout a pin can be placed inside a block. However, the pin must be connected with some pin outside the block by over-the-block routing which has to cross the block boundary. Therefore, without loss of generality, we may assume that a pin is only on the boundary of a block and can be placed in any available layer (note that stacked pins are allowed).

As we know, routing itself is an NP-hard problem. It remains NP-hard even if only two-pin nets with fixed pins between two blocks are concerned. In the traditional two-step approach, the problem is inherently difficult even for two blocks. However, the PAR

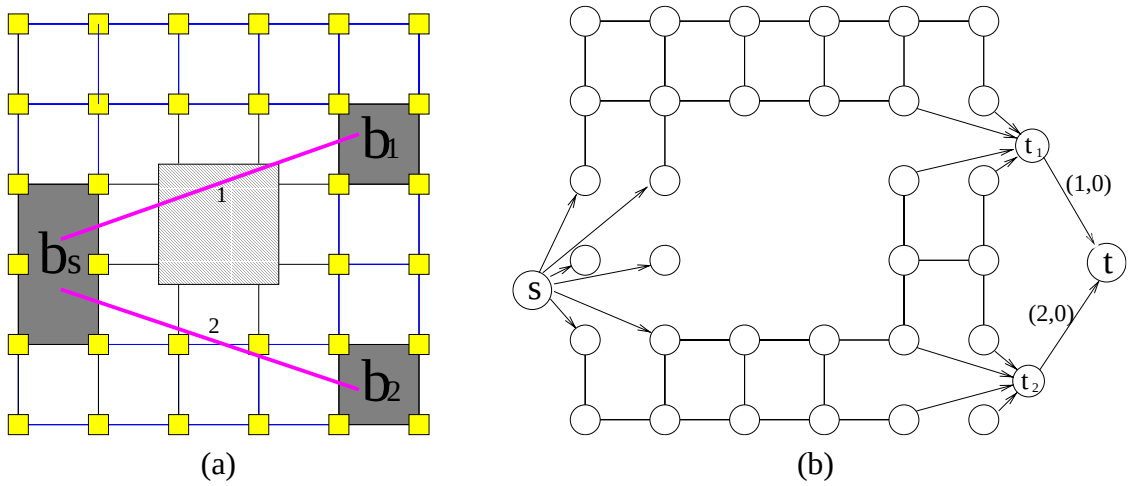


Figure 4.4 (a) A PAR problem in detailed routing. (b) The corresponding network graph.

problem, which combines pin assignment and routing, is solvable in polynomial time.

4.3 The Algorithm

In this section, we mainly use single-layer routing graph to simplify the presentation, since the single-layer illustration is easier for visualization.

To solve the PAR problem, we first construct a network graph based on the routing graph, and then apply a min-cost flow algorithm [31] to get the solution.

Given a routing grid graph $G = (V, E)$ with capacity U and cost C, blocks $B = \{b_s, b_1, b_2, ..., b_m\}$, and nets $N = N_1 \cup N_2 \cup ... \cup N_m$, the network graph $G_N = (V_N, E_N)$ is constructed as follows.

1. $V_N = \{s, t, t_1, t_2, ..., t_m\} \cup V$, where s is the source node, t is the sink node, t_i is a subsink node.

2. $E_N = E \cup \{(s, v) | v \in P_s\} \cup \{(u, t_i) | u \in P_i, i = 1, 2, ..., m\} \cup \{(t_i, t) | i = 1, 2, ..., m\}$,

where P_s is the set of the available nodes for pin assignment on the boundary of block b_s and P_i is the set of the available pin nodes on the boundary of block b_i.

3. Edge capacity: for edges (s, v) and (u, t_i), $U_N(s, v) = U_N(u, t_i) = 1$ in detailed routing and $U_N(s, v) = U_N(u, t_i) =$ pin node capacity in global routing; for edge (t_i, t), $U_N(t_i, t) = |N_i|$; for any other edge $e \in E$, $U_N(e) = U(e)$.

4. Node capacity: for $v \in V$, $U_N(v) = U(v)$. Other nodes are incapacitated.

5. Cost function: $C_N(s, v) = 0$, $C_N(u, t_i) = 0$, $C_N(t_i, t) = 0$; for other edge $e \in E$, $C_N(e) = C(e)$.

As an example, Figure 4.4(a) shows a PAR problem of detailed routing. One net is to be routed between b_s and b_1 and two nets between b_s and b_2. Figure 4.4(b) illustrates the corresponding network graph for this PAR problem. The expression (u, c) specifies the capacity u and cost c of an edge. All nodes have capacity 1. For obstacles, no edges or nodes inside obstacles are created. Therefore, no routing is inside obstacles. Note that each undirected edge represents a pair of directed edges with capacity 1 in opposite directions.

Although there are two directed edges between a pair of nodes, the two edges can not appear together in a min-cost solution and the total flow going through the two edges can not exceed the capacity of one edge. Figure 4.5 illustrates the idea. In Figure 4.5(a), suppose both edges between p and q appear in a min-cost solution. The capacity of (p, q) and (q, p) is u and the cost is c. Let (1) be the route that flow f_1 goes from s to p; let (2) be the route that flow f_1 goes from q to t; let (3) be the route that flow f_2 goes from s to q; and

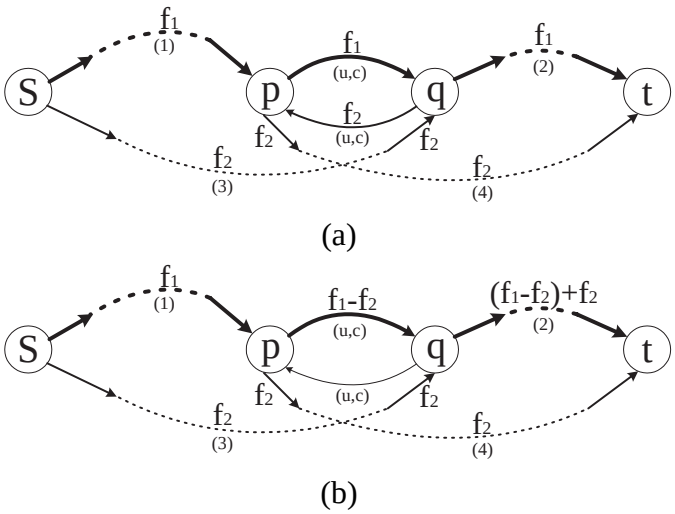


Figure 4.5 (a) A solution in a flow network. A flow f_1 goes from p to q; and another flow f_2 goes from q to p. (b) Another solution with less cost.

let (4) be the route that flow f_2 goes from p to t. The flow f_1 is $s \rightarrow (1) \rightarrow p \rightarrow q \rightarrow (2) \rightarrow t$ and f_2 is $s \rightarrow (3) \rightarrow q \rightarrow p \rightarrow (4) \rightarrow t$. ($f_1 \leq u$, and $f_2 \leq u$.) Suppose $f_1 \geq f_2$. Then the flow in Figure 4.5(a) can be transformed to the flow in Figure 4.5(b). The flow f_1 is splitted into $(f_1 - f_2)$ ($s \rightarrow (1) \rightarrow p \rightarrow q \rightarrow (2) \rightarrow t$) and f_2 ($s \rightarrow (1) \rightarrow p \rightarrow (4) \rightarrow t$); and the flow f_2 is $s \rightarrow (3) \rightarrow q \rightarrow (2) \rightarrow t$. Obviously, the total cost is reduced by $2cf_2$ since (p, q) only has a flow of $(f_1 - f_2)$. Therefore, in a min-cost flow solution, no flow goes through both edges between any pair of nodes, and the flow will not exceed the capacity of one of the two edges.

Furthermore, it is necessary to make nodes capacitated in the network graph G_N (capacitating edges only is not enough since some routes may share the same node without sharing an edge). However, classical network flow problem only capacitates edges. This can be solved by splitting the capacitated node r into two nodes r' and r'', adding an edge (r', r'') with capacity $U(r', r'') = U(r)$ and cost 0, and turning the original edges (u, r) and

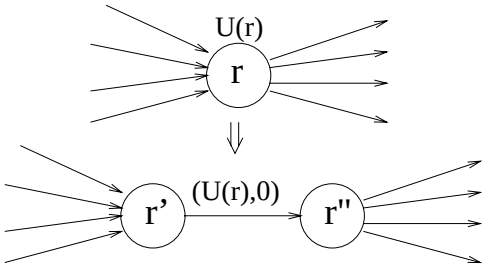


Figure 4.6 Node splitting for capacitated nodes. The capacity of the new edge is $U(r)$ and its cost is 0.

(r, v) into edges (u, r') and (r'', v) respectively (refer to Figure 4.6).

Any flow in the network G_N can be mapped to a pin assignment and routing solution for a subset of the given nets. Figure 4.7 illustrates a flow f, $|f| = 3$, corresponding to a solution of pin assignment and routing for 3 nets. Given a set of nets N, let $n = |N|$, i.e., the number of nets in N. If a flow f exists and $|f| = n$, then we can feasibly assign pins and route all nets in N. Furthermore, the cost of the flow is the cost for the solution of pin assignment and routing. Therefore, min-cost flow guarantees a solution with minimum total cost: $\alpha \cdot W + \beta \cdot V$. The total capacities of edges going into sink node t is: $\sum_{i=1}^{m} U_N(t_i, t) = \sum_{i=1}^{m} |N_i| = |N|$. Therefore, the maximum flow f_{max} in G_N, $|f_{max}| \leq |N|$. Then min-cost maximum flow assigns pins and routes for as many nets as possible with minimum total cost.

The following theorem shows that the PAR problem can be exactly solved by a min-cost flow computation on G_N.

Theorem 4.1 *A min-cost flow f, $|f| = |N|$, in G_N corresponds to a pin assignment and routing solution to PAR problem for all nets in N with minimum total cost: $\alpha \cdot W + \beta \cdot V$. If the size of the max-flow, $|f_{max}| < |N|$, then there is no feasible solution to the PAR*

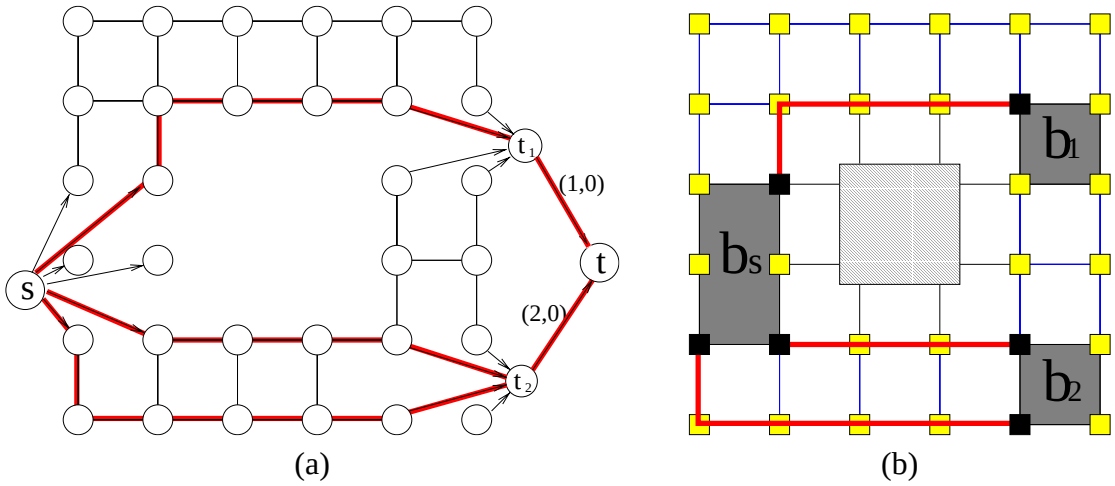


Figure 4.7 (a) A flow f in the network in Figure 4.4 (b), $|f| = 3$. (b) The corresponding solution of pin assignment and routing for the 3 nets in the problem of Figure 4.4 (a).

problem, i.e., not all nets in N are routable. A min-cost maximum flow assigns pins and routes for the maximum number of nets with minimum total cost.

Algorithm 4 summarizes the algorithm PAR-by-Flow.

Algorithm 4 PAR-by-Flow (G,U,C,B,N)

1: Construct the network graph $G_N(V_N, E_N)$

2: Assign capacities U_N and costs C_N

3: Apply min-cost max-flow algorithm on G_N

4: Derive pin assignment and routing solution

Finding a min-cost maximum flow in a network is a classical problem for which several polynomial time optimal algorithms are available [14, 15]. Deriving a solution of PAR from a flow in G_N can be done in $O(E)$ time. Thus, if we adopt the double scaling algorithm in [31], which is capable of solving integer problems, we get the following time complexity for the PAR problem.

Theorem 4.2 *The PAR-by-Flow algorithm optimally solves the PAR problem in* $O(VE \log \log U_{max} \log(VC_{max}))$ *time for* $G = (V, E)$, U_{max} *is the maximum value of* U, *and* C_{max} *is the maximum value of* C.

Note that the complexity of our algorithm PAR-by-Flow is mainly dependent on the size of the routing grid graph $G = (V, E)$. In global routing model where the size of the routing graph is smaller, PAR-by-Flow requires less runtime.

In applications, some locations on the boundary of a block may not be allowed for pin assignment. This problem can be solved easily as follows. We remove the directed edge from the source node to the boundary node of the source block, which forbids pin assignment, or remove the edge from the boundary node of the sink block to the subsink node. Then our network-flow based algorithm will not assign a pin to the location.

4.4 Applications

In this section, we discuss applications of PAR-by-Flow algorithm. PAR-by-Flow exactly solves the PAR problem, and can be used as a powerful sub-routine in many situations.

4.4.1 ECO

PAR-by-Flow provides an optimal solution to pin assignment and routing problem for all 2-pin nets from one block to other blocks. This problem matches well with some situations in ECO. Usually, a design needs to go through many changes. At each step, designers

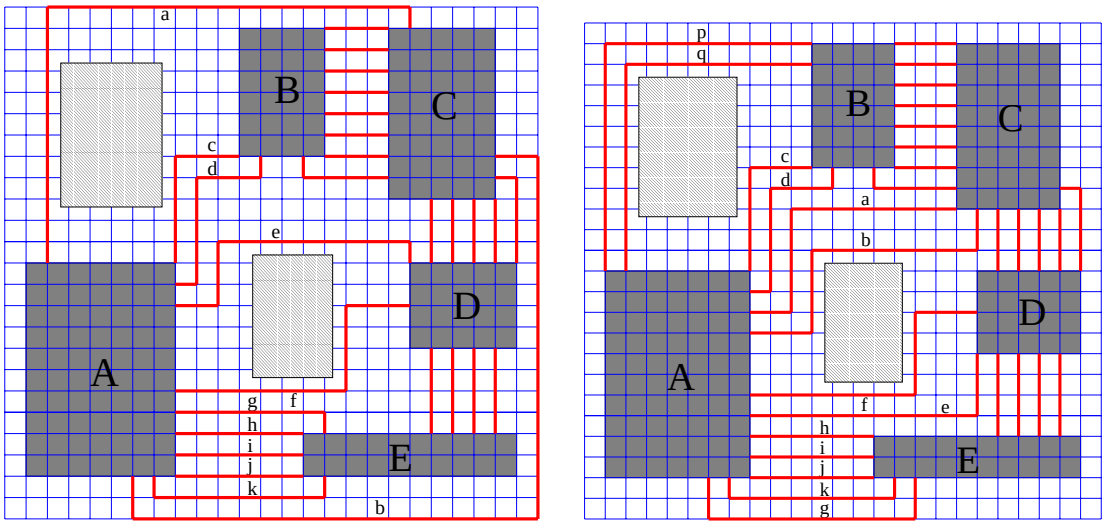


Figure 4.8 (a) The initial pin assignment and routing solution. (b) The solution obtained by applying PAR-by-Flow on Block A satisfying the new requirement.

do not want to redo everything and will just modify the existing solution incrementally. For instance, a designer changes the design of one block in a floorplan. As a result, net connections between the block and other blocks have to be changed accordingly. Some nets become unnecessary, and some new nets need to be added. Also, during rerouting, some routes are kept untouched. Now the problem becomes how to find a new solution subject to these constraints as well as minimizing the total cost $\alpha \cdot W + \beta \cdot V$. The PAR-by-Flow algorithm provides an ideal way to solve this kind of problem. For unchanged routes, we regard them as obstacles. In this way, the pins, wire segments and vias occupied by these nets can not be used by others. Then we update the set of nets according to the added or deleted nets. After removing the connections to the block, we apply PAR-by-Flow and get an optimal solution.

Figure 4.8 illustrates an example. In Figure 4.8(a), we have a pin assignment and rout-

ing design of a floorplan, and want to change net connections from Block A subject to: (1) keep the routing of 3 nets c, d, and f unchanged; (2) add two new nets p and q between A and B. Now we select Block A as the source block. Since the routes for nets c, d, f and the nets among B, C, D, and E should not be changed, they are regarded as obstacles. The set of nets becomes $\{a, b, e, g, h, i, j, k, p, q\}$. The result obtained by our algorithm is shown in Figure 4.8 (b).

4.4.2 Improvement on any given solution

Our approach has a global view of all two-pin nets connecting to one block. And it can be used to improve any pin assignment and routing solution. Given a pin-assignment and routing solution, pick a block as the source block and regard others as sink blocks, then remove all routes connected to the source block and apply PAR-by-Flow to redo pin assignment and routing. In the next step, another block is chosen as the source block and this process is repeated until all blocks are touched as the source block. The optimality of PAR-by-Flow guarantees that the new solution is no worse than the original one, either increasing the number of routed nets or reducing the cost. By repeating the procedure on each block (as source block), we can improve the solution obtained by any method. We call this iterative method as IMProve-by-PAR.

As we notice, the ordering of source blocks in IMProve-by-PAR has influence on the final result. Different orderings may lead to different results since each step is based on the previous step. To alleviate the influence of block order, we implement IMProve-by-PAR

by enforcing a random order on blocks to apply PAR-by-Flow. Furthermore, we repeat IMProve-by-PAR several times to get a better result. Each time, we get a new solution from IMProve-by-PAR, and let this new solution be the input of the next IMProve-by-PAR.

This repeated application of IMProve-by-PAR is referred to RepIMProve-by-PAR, and its pseudocode is shown in *Algorithm* 5. S is a previous solution, and T is the iteration times.

Algorithm 5 RepIMProve-by-PAR (G,U,C,N,B,S,T)

for $i = 1$ to T **do**

Generate a random order $Order$ on blocks B

S = IMProve-by-PAR (G,U,C,N,B,S, $Order$)

end for

Figure 4.9 illustrates an example of IMProve-by-PAR. The numbers in Figure 4.9(a) are the number of nets to be routed between two blocks. The target is to route 1 net between A and B, A and C, 4 nets between A and D, and 3 nets between B and C, B and D, C and D. Figure 4.9(b) illustrates the initial net-by-net solution for pin assignment and routing among the 4 blocks based on the min-cost path approach. In this solution, only 10 nets are routed, and 5 nets (2 nets between blocks B and C; 3 nets between blocks C and D) are not routed. The total cost is 51. First we choose Block A as the source block. After removing all routes connected to Block A, we apply PAR-by-Flow to find net connections for Block A and get a new solution as shown in Figure 4.9(c). In this step, the cost is reduced by 15. Then we apply PAR-by-Flow to Block B as Figure 4.9(d). Two more nets between B and C are routed. Similarly, Figure 4.9(e) shows the solution after applying PAR-by-Flow on

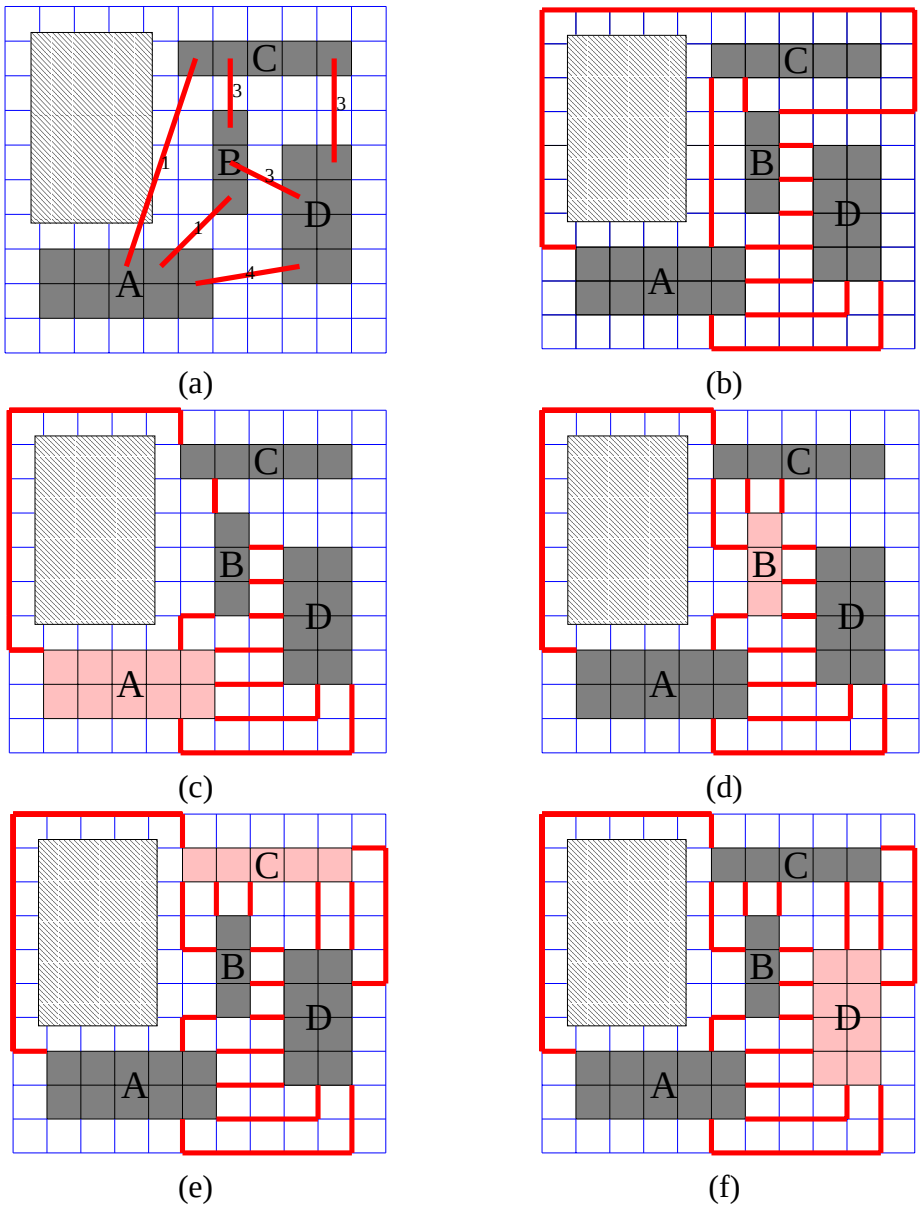


Figure 4.9 Illustration of improvement on a given solution. (a) Illustration of net connections among Block A, B, C and D. (b) Initial net-by-net solution based on the min-cost path approach. 5 nets (2 between B and C; 3 between C and D) are not routed. The total cost is 51. (c) The solution after applying PAR-by-Flow on Block A. Cost is reduced by 15. (d) The solution after applying PAR-by-Flow on Block B. Two more nets between B and C are routed. (e) The solution after applying PAR-by-Flow on Block C. All nets are routed (3 more nets) with less cost (from 51 to 50). (f) The solution after applying PAR-by-Flow on Block D. Nothing is changed.

Block C. All nets are routed (3 more nets) with less cost (from 51 to 50). Finally, apply PAR-by-Flow on Block D and get the complete solution as Figure 4.9(f). Note that for the final solution, the total cost is reduced from 51 to 50 even though 5 more nets are routed.

In addition, IMProve-by-PAR itself provides a new way to solve pin assignment and routing problem among multiple macro blocks. The general problem can be decomposed to a set of PAR problems and solved by PAR-by-Flow iteratively. Again, to alleviate the influence of block order, we just choose source block randomly. This comes out a polynomial-time randomized algorithm to solve the pin assignment and routing problem among blocks. Of course, RepIMProve-by-PAR can be used to improve the result. In fact, if we just let the input solutions of RepIMProve-by-PAR S empty, we can always get one solution when the program terminates.

4.4.3 Multiple-pin nets

In the above sections, we focus on two-pin net connections among multiple macro blocks. Now we extend the algorithm to handle both two-pin nets and multiple-pin nets.

Suppose a multiple pin net is routed as a steiner tree. Once a block is selected as the source block, the branch on the tree connecting to the source block is removed to reroute. Since the routing of the rest of the tree (called residual tree) should not be changed, the edges and nodes occupied by the residual tree are treated as obstacles. Furthermore, one new node v is added to the flow network to present this multiple pin net. All nodes on the residual tree are connected to v and v is connected to the sink node t. The capacity of the

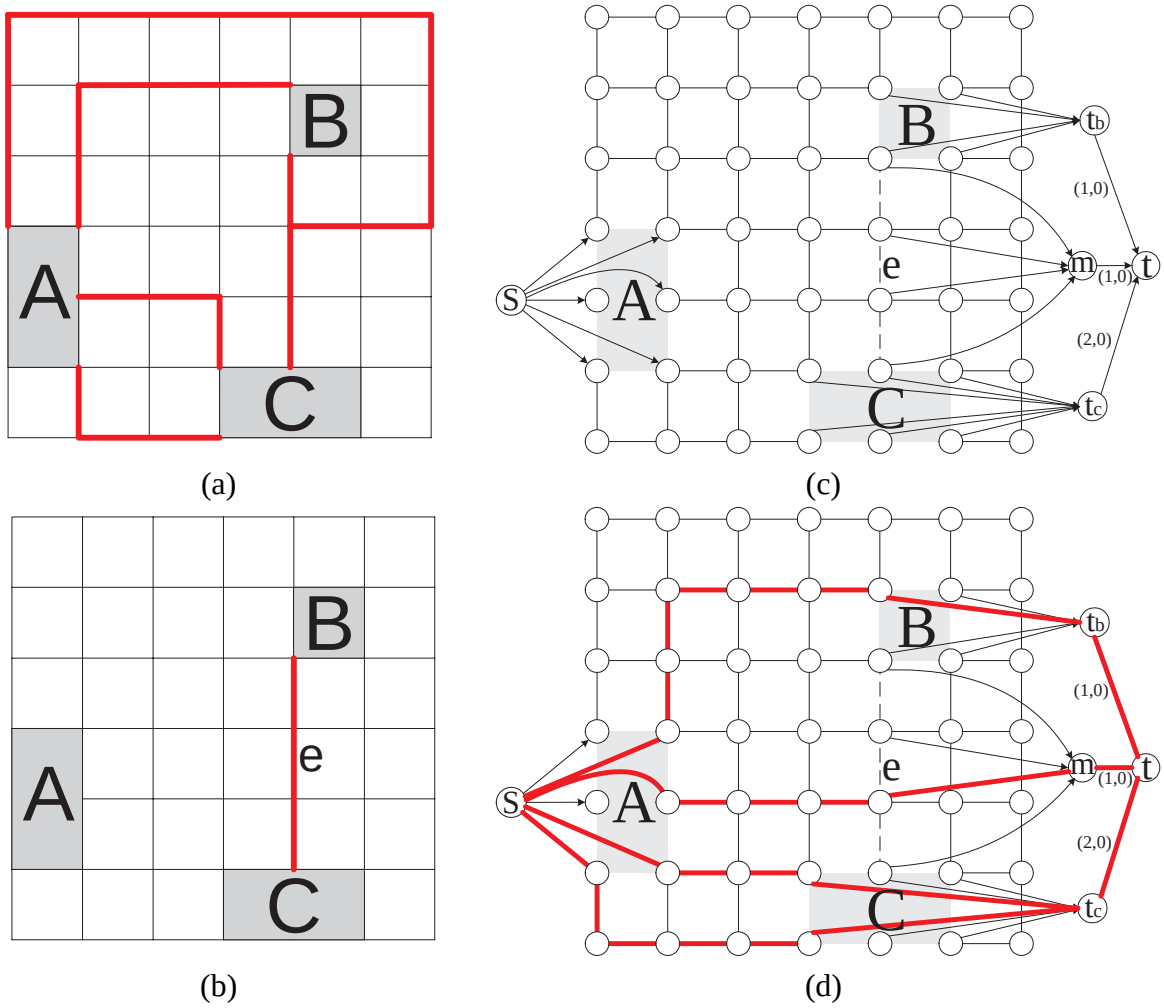


Figure 4.10 Illustration of improvement for a pin assignment and routing of two/multiple-pin nets. (a) A one-layer pin-assignment and routing solution. (b) When Block A is selected to be the source block, all nets connecting to A are removed to reroute. The routing e between B and C should not be changed. (c) The corresponding flow network graph. (d) A flow f ($|f| = 4$) in the network.

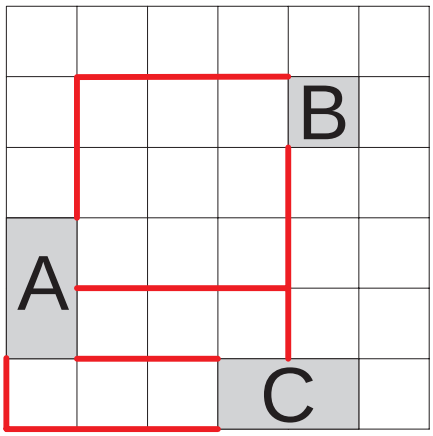


Figure 4.11 An improved solution of pin assignment and routing of two/multiple-pin nets.

edge (v, t) is 1 and the cost is 0. Therefore only one flow is allowed from v to t, i.e., only one node on the residual tree is connected to the source block.

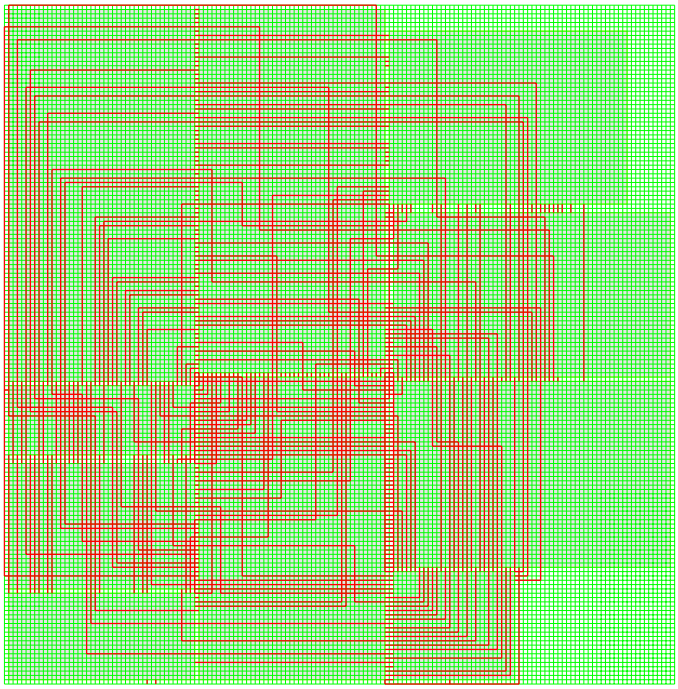
Figure 4.10 illustrates an example. Figure 4.10(a) shows the initial solution of pin assignment and routing for a one layer problem. When Block A is selected, all routed nets connecting to A are removed for rerouting as illustrated in Figure 4.10(b). And the routing e between B and C should not be changed. We construct the flow network as described in section 4.3. For the multiple pin net among blocks A, B and C, a new node m is created to represent this net. All nodes along e are connected to m and m is connected to the sink node t. Furthermore, the edges along e have already been occupied. Therefore the edges along e are deleted (represented by dash line in Figure 4.10(c) and (d)), and no more flows can push through them. The corresponding flow network is shown in Figure 4.10(c). Figure 4.10(d) shows a flow f ($|f| = 4$) in the network. By mapping the flow to the original routing grid, we get a new pin assignment and routing solution as Figure 4.11. Compared to the original one, the total wire length is reduced from 28 to 17.

Moreover, the above method is also a way to solve pin assignment and routing problem of two/multiple pin nets. For each multiple-pin net, we can first set up one connection between two blocks (just as a two-pin net between the two blocks). Connecting all nodes on the tree(line) to node v which presents the multiple-pin net, we can construct a steiner tree by adding branches one by one. Once every block has been selected as the source block, we can get a pin assignment and routing solution for nets among multiple blocks. Still we can further improve the solution by repeating the above procedure several times. A different block ordering can be adopted each time.

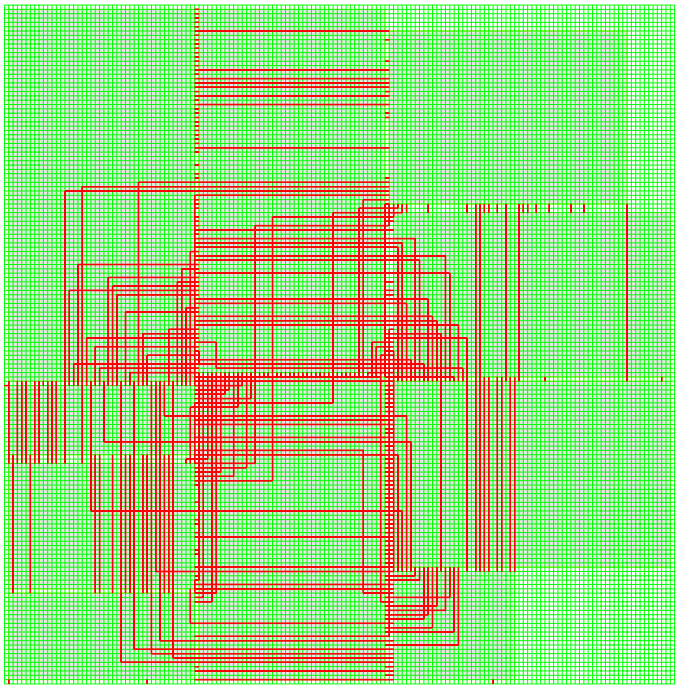
4.5 Experimental Results

We have implemented the PAR-by-Flow and RepIMProve-by-PAR algorithms in C++ language, and carried out experiments on Sun Sparc Ultra 5 (360MHz) with 128MB memory.

We have tested the refinement algorithm RepIMProve-by-PAR for two pin nets by using a net-by-net approach to get the original solution. The net-by-net approach considers only one net each time, and finds a min-cost path between source block and sink blocks to assign pins and route the net. Nets in net-by-net approach are processed randomly. In RepIMProve-by-PAR, we repeat IMProve-by-PAR 10 times. Both RepIMProve-by-PAR and net-by-net are executed 10 times on 8 data files, 4 for detailed routing and 4 for global routing. Tables 4.1 and 4.2 list the average of these ten results for detailed routing and global routing respectively. After refinement, all nets are routed with significant improvement on the total cost, the wire length and the number of vias. As an illustration, Figure



(a)



(b)

Figure 4.12 Two-layer pin assignment and routing for X18. (a) Net-by-net solution. (b) The solution obtained by applying our method on (a).

Table 4.1 Average results of 10 times for detailed routing test files. All nets are routed after refinement by RepIMProve-by-PAR.

Detailed Routing					
File		C2	Am3	P2	X18
Grid		67x61	134x140	118x108	155x157
Block		11	33	12	10
Node		8458	38140	25970	49320
Edge		29992	125956	86655	157119
Capacity		1	1	1	1
Net		152	355	295	268
Time (sec)	Previous	3.78	35.42	16.44	32.06
	Refined (per iter)	8.94	136.76	43.03	101.72
Routed Nets	Previous	150.2	354.5	293.6	263.9
	Refined	152	355	295	268
Cost	Previous (per net)	2575.0 (17.14)	16215.7 (45.74)	6093.8 (20.76)	9817.4 (37.20)
	Refined (per net)	1966.0 (12.93)	14597.6 (41.12)	5154.5 (17.47)	7191.0 (26.83)
	Improve	24.6%	10.1%	15.8%	27.9%
Wire	Previous (per net)	2502.3 (16.66)	15906.8 (44.87)	6012.9 (20.48)	9659.8 (36.60)
	Refined (per net)	1909.9 (12.57)	14362.7 (40.46)	5098.0 (17.28)	7081.9 (26.43)
	Improve	24.5%	9.8%	15.6%	27.8%
Via	Previous (per net)	72.7 (0.48)	308.9 (0.87)	80.9 (0.28)	157.6 (0.60)
	Refined (per net)	56.1 (0.37)	234.9 (0.66)	56.5 (0.19)	109.1 (0.41)
	Improve	22.9%	24.1%	32.1%	31.7%

Table 4.2 Average results of 10 times for global routing test files. All nets are routed after refinement by RepIMProve-by-PAR.

Global Routing					
File		K1	V2	Z2	N3
Grid		20x21	41x44	33x30	50x47
Block		8	20	28	40
Node		944	3824	2168	4980
Edge		3695	14307	8615	18987
Capacity		55	30	70	25
Net		2131	3088	5498	2588
Time (sec)	Previous	5.15	27.52	34.83	43.31
	Refined (per iter)	2.48	26.62	15.57	54.17
Routed Nets	Previous	2117.9	3050.1	5440.1	2572.8
	Refined	2131	3088	5498	2588
Cost	Previous (per net)	17787.6 (8.40)	52744.8 (17.29)	73015.8 (13.42)	63484.7 (24.68)
	Refined (per net)	14966.3 (7.02)	48113.8 (15.58)	66722.2 (12.14)	57308.2 (22.14)
	Improve	16.4%	9.9%	9.5%	10.3%
Wire	Previous (per net)	16295.7 (7.69)	50782.4 (16.65)	68586.1 (12.61)	60861.0 (23.66)
	Refined (per net)	13728.9 (6.44)	46611.0 (15.09)	63031.3 (11.46)	55172.4 (21.32)
	Improve	16.3%	9.4%	9.1%	9.9%
Via	Previous (per net)	1491.9 (0.70)	1962.4 (0.64)	4429.7 (0.81)	2623.7 (1.02)
	Refined (per net)	1237.4 (0.58)	1502.8 (0.49)	3690.9 (0.67)	2135.8 (0.83)
	Improve	17.1%	23.4%	17.3%	18.6%

4.12 shows the results of pin assignment and routing for input file "X18", where $\alpha = 1$ and $\beta = 1$. Vertical tracks are in Layer 1 and horizontal tracks are in Layer 2. Figure 4.12(a) shows the net-by-net solution. Only 262 nets are routed and the total cost is 11023. Figure 4.12(b) is obtained by applying our method on (a). All nets (totally 268) are routed, and the total cost is 7153.

4.6 Conclusion

In this chapter, we considered the problem of two-pin net connections from one block to all other blocks. We presented the first polynomial-time optimal algorithm for simultaneous pin assignment and routing in multilayer for all two-pin nets between a source block and all other blocks. Our algorithm is applicable for both global routing and detailed routing with arbitrary routing obstacles on multiple layers, and guarantees a pin-assignment and routing solution with minimum total cost $\alpha \cdot W + \beta \cdot V$ by computing a min-cost flow in a network. In an ECO situation where a designer does not want to redo everything after a change, the algorithm provides an ideal way for incremental modification of the existing solution. Also, it can be applied to improve any pin assignment and routing solution by any existing method. Furthermore, by applying the algorithm iteratively for all blocks (each time randomly pick one block as the source block), it provides a polynomial-time randomized algorithm to solve the pin assignment and routing problem of all blocks. Experiments demonstrate that the algorithm is very efficient and effective.

CHAPTER 5

INTEGRATED PIN ASSIGNMENT AND BUFFER PLANNING

5.1 Introduction

As chip size grows larger and minimum feature size is reduced, the capacitance and resistance of wires increase dramatically. This makes interconnects play a critical role in achieving high performance and reducing circuit complexity in deep-submicron design. Many techniques have been proposed to reduce interconnect delay. One effective way is buffer insertion [32, 33] and it is heavily used in chip design. It is estimated that the amount of inserted buffers for on-chip interconnect may be up to $800,000$ in $50nm$ technology [34]. At the same time, the introduction of so many buffers raises many challenging problems in physical design. In a hierarchical approach, buffers are clustered together as a buffer block to facilitate floorplan and routing. Cong et al. [35], Tang and Wong [36] and Sarkar et al. [37] proposed algorithms for buffer block planning problem to minimize the chip area and the number of buffer blocks. In two recent works, Dragan et al. [38, 39] gave algorithms for global buffered routing problem with fixed pins. Their work is based on multicommodity

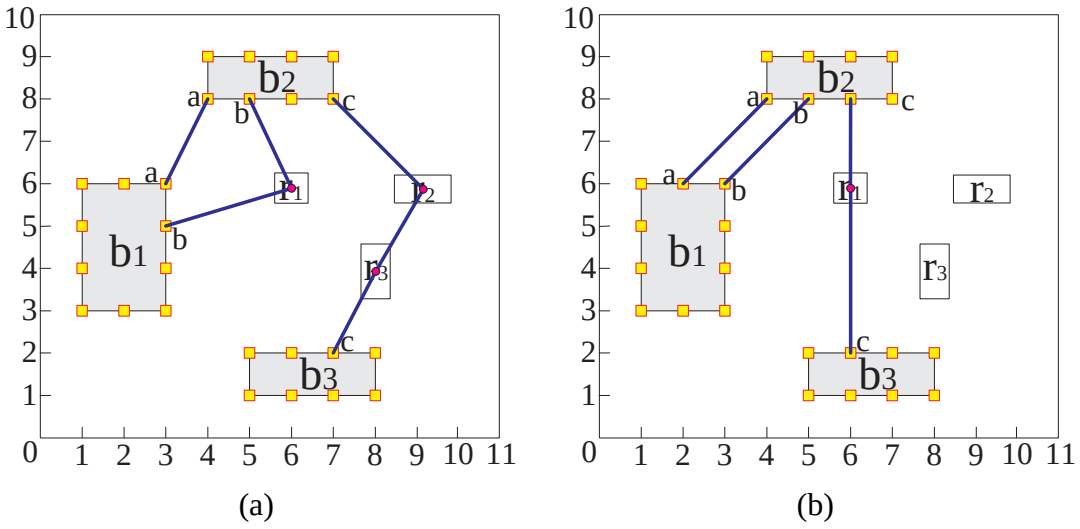


Figure 5.1 (a) Three nets use 3 buffers and the total wire length is 19. (b) An optimal solution with 1 buffer and wire length 14.

flow, which is an NP-hard problem. In this chapter, we address the problem of simultaneous Pin assignment and Buffer planning (PB) for a given buffer block plan. Our algorithm uses min-cost flow computation which is solvable in polynomial time.

Informally, the problem can be described as follows: given a placement of macro blocks and buffer blocks, assign pins and plan buffer insertions for the given set of nets subject to the required lower and upper bounds of connection intervals (i.e., the range of allowable distance between two buffers or two pins or a pin and a buffer) as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$ where W is the total wire length and R is the number of buffers. Figure 5.1 shows an example. The placement includes three macro blocks, and three buffer blocks. Buffer blocks may have different capacities; i.e., the number of buffers in a buffer block can be different. r_1 has a capacity 1 while the capacities of r_2 and r_3 are 2. A net set includes 2 nets between b_1 and b_2 and 1 net between b_1 and b_3. The range of allowable distance between two buffers or a buffer and a pin is bounded by Manhattan distance 2

and 4. Also if the distance of two pins is longer than 5, buffers have to be inserted; i.e., the lower and upper bounds for the allowable distance between two pins are 0 and 5. The purpose is to assign pins and plan buffers for the 3 nets while minimizing the number of buffers and the total wire length.

The goal of pin assignment is to find the exact locations of pins on macro blocks. Buffer planning is to decide buffer usages along net connections to maintain required delay constraints. The two tasks are closely related. Pin assignment alone may neglect many important factors since interconnect is hard to predict, and a global view of net connections is always helpful. Figure 5.1(a) illustrates a solution by a two-step approach (i.e., pin assignment following buffer planning). The pin assignments are decided according to the shortest Manhattan distance. In total, three buffers are used and the wire length is 19. Figure 5.1(b) shows an optimal solution with only one buffer and the wire length is 14.

In this chapter, we present a polynomial-time exact algorithm for simultaneous pin assignment and buffer planning for all two-pin nets from one macro block (source block) to all other blocks for a given buffer block plan such that each net satisfies the lower and upper bounds on connection intervals as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$ for any positive constants α and β where W is the total wire length and R is the number of buffers. By applying this algorithm iteratively (each time pick one block as the source block), it provides a polynomial-time algorithm for pin assignment and buffer planning for nets among multiple macro blocks. Experimental results demonstrate its efficiency and effectiveness.

The rest of this chapter is organized as follows. Section 5.2 defines the PBO (Pin assignment and Buffer planning for One source block) problem which simultaneously assigns pins and plans buffers for nets between one macro block and all other blocks. In Section 5.3, we present a network flow formulation to solve the PBO problem. Then we extend PBO problem to PB (Pin assignment and Buffer Planning) problem that considers all nets among multiple macro blocks in Section 5.4. In Section 5.5, we also provide a node clustering method to speed up the computation. Finally, we show the experimental results in Section 5.6 and conclude the paper in Section 5.7.

5.2 Pin Assignment and Buffer Planning for One Source Block (PBO)

Given a placement of $m+1$ macro blocks $B = \{b_s, b_1, ..., b_m\}$ with n buffer blocks $R = \{r_1, ..., r_n\}$. (For convenience, we call b_s the source block and other blocks sink blocks.) Each buffer block r_i $(i = 1, ..., n)$ is associated with a positive integer c_i denoting the capacity of r_i; i.e., buffer block r_i can hold c_i buffers.

Let $N = N_1 \cup N_2 \cup ... \cup N_m$ where N_i $(i = 1, ..., m)$ is the set of nets between block b_s and b_i; and $P = P_s \cup P_1 \cup ... \cup P_m$ where P_i $(i = s, 1, ..., m)$ is the set of available pin locations of block b_i.

The distance of two points u and v on a planar region is denoted as d_{uv}. Let *buffer interval* be the allowable distance range between two buffers or a pin and a buffer; and *pin*

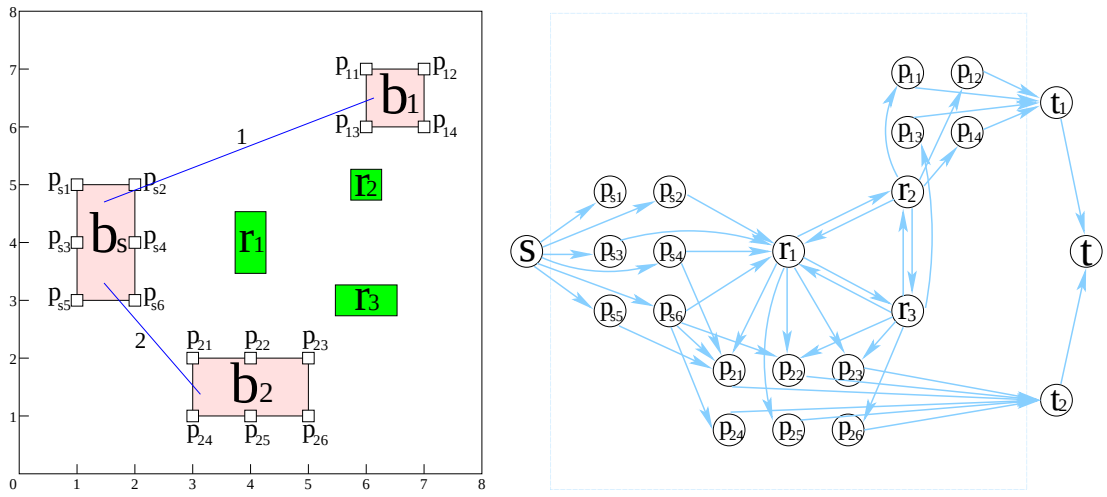


Figure 5.2 (a) A PBO problem with 3 macro blocks and 3 buffer blocks. (b) The corresponding flow network graph.

interval be the allowable distance range between two pins. For convenience, let *connection interval* refer to buffer interval or pin interval.

Suppose the lower and upper bounds for buffer intervals are $\tilde{L}$ and $\tilde{U}$ respectively; and those for pin intervals are $\bar{L}$ and $\bar{U}$ respectively. A valid path means:

1. If the path is $p = (p_s, r'_1, ..., r'_k, p_t)$ where $p_s, p_t \in P$ and $r'_i \in R$ $(i = 1, ..., k)$, then the distance of each path segment is bounded by $\tilde{L}$ and $\tilde{U}$, i.e., $\tilde{L} \leq d_{p_s r'_1} \leq \tilde{U}$, $\tilde{L} \leq d_{r'_i r'_{i+1}} \leq \tilde{U}$ $(i = 1, ..., k-1)$ and $\tilde{L} \leq d_{r'_k p_t} \leq \tilde{U}$;

2. If the path is $p = (p_s, p_t)$ where $p_s, p_t \in P$, then $\bar{L} \leq d_{p_s p_t} \leq \bar{U}$.

The length of a path is the sum of the distances of all path segments.

For any given positive constants α and β, the PBO problem is to find a set of valid paths connecting b_s and all other macro blocks as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$ where W is the total length and R is the number of buffers. Each path corresponds to a net

in N and the two end points of the path are assigned pin locations for this net.

Figure 5.2(a) gives a simple example. There are three macro blocks and three buffer blocks. The capacities of the three buffer blocks r_1, r_2, and r_3 are 2, 1, and 2, respectively. For any two points u and v on the planar region, their coordinates are (u_x, u_y) and (v_x, v_y), respectively. Define distance $d_{uv} = |u_x - v_x| + |u_y - v_y|$. The required lower and upper bounds for buffer intervals are 2 and 3, respectively; and the bounds for pin intervals are 0 and 3, respectively. The tiny squares p_{xy} on the boundaries of macro blocks are available pin locations. The net set includes one net between b_s and b_1 and two between b_s and b_2. The purpose is to decide pins and buffer locations for 1 net between b_s and b_1 and 2 nets between b_s and b_2 with a minimum cost $\alpha \cdot W + \beta \cdot R$.

5.3 The Algorithm

To solve the PBO problem, we first construct a network graph, then apply a min-cost flow algorithm to get the solution.

Given a PBO problem, we construct the network graph $G = (V, E)$ with capacity U and cost C as follows:

1. $V = \{s, t, t_1, t_2, ..., t_m\} \cup R \cup P$, where s is the source node, t is the sink node, and t_i $(i = 1, ..., m)$ is a subsink node.

2. $E = E_s \cup E_t \cup E_{\tilde{t}} \cup E_b \cup E_r \cup E_{br} \cup E_{rb}$, where

$E_s = \{(s, p_s)|p_s \in P_s\}$,

$E_t = \{(t_i, t)|i = 1, 2, ..., m\}$,

$E_{\tilde{t}} = \bigcup_{i=1}^{m}\{(p, t_i)|p \in P_i\}$,

$E_b = \bigcup_{i=1}^{m}\{(p_s, p)|p_s \in P_s, p \in P_i, \bar{L} \leq d_{p_s p} \leq \bar{U}\}$,

$E_r = \{(r_i, r_j)|i \neq j, i, j = 1, ..., n, \tilde{L} \leq d_{r_i r_j} \leq \tilde{U}\}$,

$E_{br} = \bigcup_{i=1}^{n}\{(p_s, r_i)|p_s \in P_s, \tilde{L} \leq d_{p_s r_i} \leq \tilde{U}\}$,

$E_{rb} = \bigcup_{i=1}^{n}\{(r_i, p)|p \in P_j, j = 1, ..., m, \tilde{L} \leq d_{r_i p} \leq \tilde{U}\}$

3. Edge capacity:

   for edges $e(t_i, t)$, $U(e) = |N_i|$,$(i = 1, ..., m)$;

   other edges e, $U(e) = 1$.

4. Node capacity:

   for $r_i \in R$, $U(r_i) = c_i$;

   for $p \in P$, $U(p) = 1$;

   other nodes are incapacitated.

5. Edge cost:

   for $e \in E_s \cup E_t \cup E_{\tilde{t}}$, $C(e) = 0$;

   other edges $e(u, v)$, $C(e) = \alpha \cdot d_{uv}$.

6. Node cost:

   for $r \in R$, $C(r) = \beta$;

   other nodes v, $C(v) = 0$.

Figure 5.2(b) illustrates the constructed network graph for the PBO problem in Figure 5.2(a). Note that whether an edge (u, v) $(u, v \in P \cup R)$ should be added to G depends on

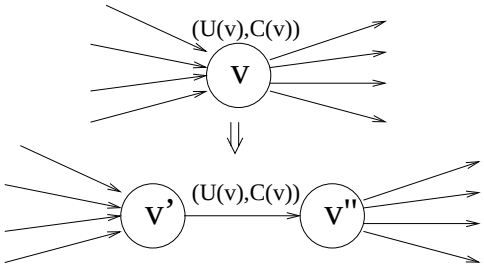


Figure 5.3 Node splitting for capacitated nodes. The new edge has capacity $U(v)$ and cost $C(v)$.

whether d_{uv} falls in the range $[\tilde{L}, \tilde{U}]$ (or $[\bar{L}, \bar{U}]$) or not. For example, the distance between p_{13} and r_2 is 1, which is less than the lower buffer interval bound $\tilde{L} = 2$. Thus, there is no edge between p_{13} and r_2 in the constructed flow network. Similarly, the distance between p_{s2} and p_{13} is 5 which is larger than the upper pin interval bound $\bar{U} = 3$, thus the edge (p_{s2}, p_{13}) is not included.

In the constructed flow network, every node in $P \cup R$ has a cost and a capacity. However, classical network flow problem only assigns cost and capacity to edges. This can be solved by splitting the capacitated node v into two nodes v' and v''. A new edge (v', v'') is added with a capacity $U(v)$ and a cost $C(v)$. Then change the original edges (u, v) and (v, w) into edges (u, v') and (v'', w) respectively (refer to Figure 5.3).

Any flow in G can be mapped to a pin assignment and buffer planning solution for a subset of the given nets. The used capacities of R nodes in the flow are the number of buffers needed in the solution. Figure 5.4(a) illustrates a flow f, $|f| = 3$. And Figure 5.4(b) is a solution of pin assignment and buffer planning derived from the above flow. If a flow f exists and $|f| = |N|$, then we can find a feasible solution of pin assignment and buffer planning for all of the nets in N. On the other hand, given a pin assignment

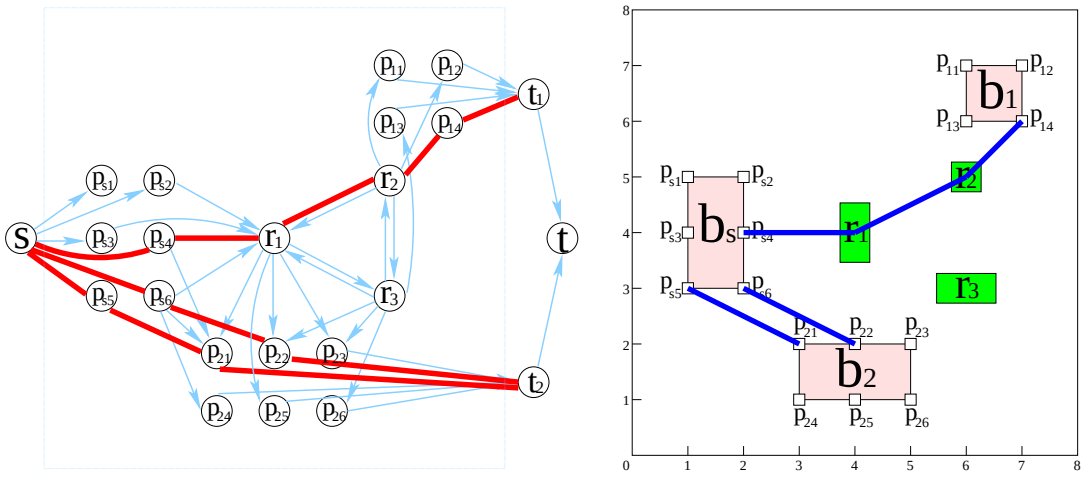


Figure 5.4 (a) A flow f in the network in Figure 5.2 (b), $|f| = 3$. (b) The corresponding solution of pin assignment and buffer planning to the PBO problem of Figure 5.2 (a).

and buffer planning solution for n nets, a flow f ($|f| = n$) can always be found on the constructed flow network. Since the total capacities of edges going into sink node t are $\sum_{i=1}^{m} U(t_i, t) = \sum_{i=1}^{m} |N_i| = |N|$, the maximum flow f_{max} in G, $|f_{max}| \leq |N|$. Thus if a flow $|f| < |N|$, then there is no feasible solution to the original PBO problem, which requires considering all of the nets between b_s and all other macro blocks. Furthermore, the cost of the flow is also the cost of pin assignment and buffer planning solution. Therefore min-cost maximum flow assigns pins and plans buffers for as many nets as possible with minimum total cost.

The following theorem shows that the PBO problem can be exactly solved by min-cost flow computation on G.

Theorem 5.1 *A min-cost flow f, $|f| = |N|$, in G corresponds to a pin assignment and buffer planning solution to PBO problem for all nets in N with minimum total cost $\alpha \cdot W + \beta \cdot R$ for any given α and β where W is the total wire length and R is the number of*

buffers. If the size of the max-flow, $|f_{max}| < |N|$, then there is no feasible solution to the PBO problem. A min-cost maximum flow assigns pins and plans buffers for the maximum number of nets with minimum total cost.

The algorithm PBO-Flow is summarized in *Algorithm* 6.

Algorithm 6 PBO-Flow (B, R, N, P, C, α, β)

1: Construct the network graph $G(V, E)$
2: Assign capacities U and costs C
3: Apply min-cost maximum flow algorithm on G
4: Derive the pin assignment and buffer planning solution

Finding a min-cost maximum flow in a network is a classical problem for which several polynomial-time optimal algorithms are available [14, 15]. Deriving a solution of PBO from a flow in G can be done in $O(E)$ time. Thus, if we adopt the double scaling algorithm in [31], we get the following time complexity for the PBO problem.

Theorem 5.2 *PBO-Flow algorithm optimally solves the PBO problem in $O(VE \log\log U_{max} \log(VC_{max}))$ time for $G = (V, E)$, U_{max} is the maximum value of U, and C_{max} is the maximum value of C.*

Note that the complexity of our PBO-Flow algorithm is mainly dependent on the size of the constructed network graph $G(V, E)$. According to the graph construction rules, $|V| = 2 + m + 2(|P| + |R|)$ (consider node splitting), and the upper bound of edges is $m + |P| + |P_s| \cdot |P| + |P| \cdot |R| + |R|^2 + |P| + |R|$ (in fact, the number of edges is much smaller due to distance constraint) which is bounded by $(|P| + |R|)^2$.

In applications, we may put more effort on reducing the number of buffers. In this case, we can set a large weight to buffer nodes, i.e., a large β. For example, let β larger than the upper distance bound $\tilde{U}$. If we set β large enough, we tend to get a solution with the minimum number of buffers.

Furthermore in some circuits, some locations on a block may not be allowed for pin assignment. In this case, these kinds of locations will not appear in the pin set P. Obviously, our network-flow based algorithm will not assign pins to these kinds of locations.

5.4 Pin Assignment and Buffer Planning (PB)

In the above section, we discuss how to solve PBO problem using min-cost maximum flow computation. PBO problem only consider net connections between one source block and other blocks. In reality, we need to deal with net connections among all of the macro blocks, called PB problem. The definition of PBO problem can be easily extended to PB problem as the following:

Given:

1. A placement of m macro blocks $B = \{b_1, b_2, ..., b_m\}$ and n buffer blocks $R = \{r_1, r_2, ..., r_n\}$; buffer block r_i has a capacity c_i.

2. A set of available pin locations $P = P_1 \cup P_2 \cup ... \cup P_m$ where P_i $(i = 1, ..., m)$ is a set of available pin locations of macro block b_i.

3. A set of nets $\bar{N} = \bar{N}_1 \cup \bar{N}_2 \cup ... \cup \bar{N}_m$ where $\bar{N}_i$ $(i = 1, ..., m)$ is the set of nets

between block b_i and all other blocks.

4. Two nonnegative numbers $\tilde{L}$ and $\tilde{U}$ denoting the lower and upper bound of buffer intervals, respectively.

5. Two nonnegative numbers $\bar{L}$ and $\bar{U}$ denoting the lower and upper bound of pin intervals, respectively.

Goal:

For any given positive α and β, find a set of valid paths corresponding to the net set $\bar{N}$ as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$ where W is the total wire length and R is the number of buffers.

PBO-Flow algorithm solves pin assignment and buffer planning for nets between one block and all other blocks. Naturally, if we treat each macro block as the source block and apply PBO-Flow algorithm repeatedly, we can get a solution for all nets among multiple macro blocks. *Algorithm* 7 shows the basic idea of the PB-Flow algorithm.

Algorithm 7 PB-Flow (B, R, $\bar{N}$, P, C, α, β)

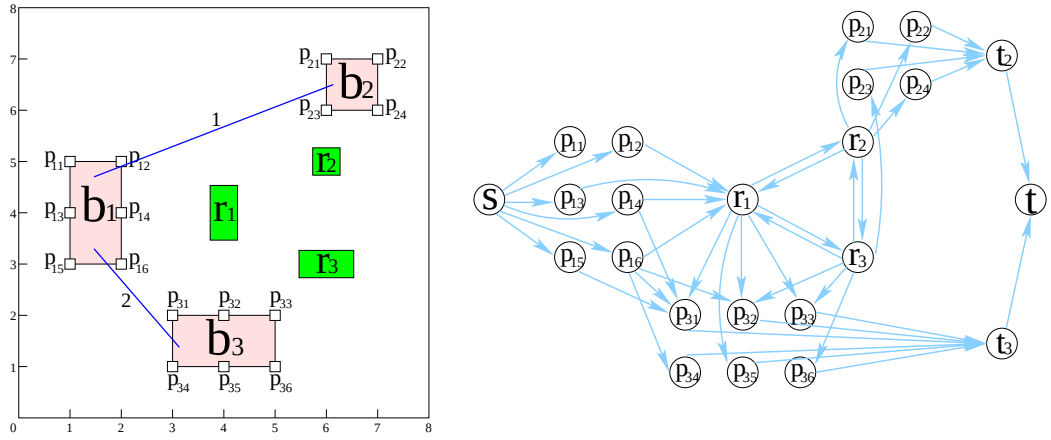
1: Let b_1 be the source block
2: Apply PBO-Flow(B, R, $\bar{N}_1$, P, C, α, β)
3: **for** $i = 2$ to m **do**
4: Let b_i be the source block
5: Adjust the network graph $G(V, E)$
6: Apply PBO-Flow(B, R, $\bar{N}_i$, P, C, α, β)
7: **end for**
8: Derive the pin assignment and buffer planning solution

Note that when different blocks are selected as the source block, their constructed flow networks are slightly different, i.e., the edges incident from/to the pin nodes of the two blocks are different. So in PB-Flow algorithm, at each iteration, we need to do some adjustment to transform the existing network graph to the one corresponding to the next source block (line 6).

Now suppose two different macro blocks b_i and b_{i+1} are selected as the source block sequentially. Let G_i be the constructed flow network when b_i is chosen as the source block. The following steps change G_i to b_{i+1}'s corresponding flow network G_{i+1}:

1. Delete edges connecting the pin nodes of P_i and the pin nodes of other blocks.
2. Add edges connecting the pin nodes of P_{i+1} and the pin nodes of other blocks as well as maintaining distance constraints.
3. Reverse the direction of edges (p, r) where $p \in P_i$ and $r \in R$;
4. reverse the direction of edges (r, p) where $r \in R$ and $p \in P_{i+1}$.
5. Reverse the direction of edges (s, p) where $p \in P_i$.
6. Reverse the direction of edges (p, t_{i+1}) where $p \in P_{i+1}$.
7. Let t_{i+1} be the source node s and the original source node become a subsink node t_i. Thus remove the edge (t_{i+1}, t) and add one new edge (t_i, t).

Figure 5.5(a) is a PB problem. Figure 5.5(b) shows the constructed flow network when b_1 is the source block. When the source block is switched to b_2, t_2 becomes the source node



(a) (b)

Figure 5.5 (a) A PB problem with 3 macro blocks and 3 buffer blocks. (b) The corresponding flow network when b_1 is the source block.

and the original one becomes a subsink node t_1, but the edges connecting two buffer nodes remain unchanged.

For a newly added edge (u, v), the cost is $\alpha \cdot d_{uv}$, and the capacity is 1. For other edges (u', v'), the cost is unchanged and the capacity is 1 if no flow flows through the corresponding edge in the previous iteration. Otherwise, the capacity is 0. As to the capacities of nodes, similarly, if a node v has c capacities left after pushing flow in G_i, then the capacity of v is c in G_{i+1}.

G_i and G_{i+1} have the same node set. Also, it is easy to show that the number of changed edges is bounded by $(|P_i| + |P_{i+1}|) \cdot (|P| + |R|)$. When taking the distance constraint into consideration, this bound can be further greatly reduced. Thus the adjustment to the graph can be efficiently accomplished.

A two-pin net connecting b_i and b_{i+1} belongs to both net sets $\bar{N}_i$ and $\bar{N}_{i+1}$. Suppose we can get a feasible solution to the PB problem with PB-Flow algorithm. After applying

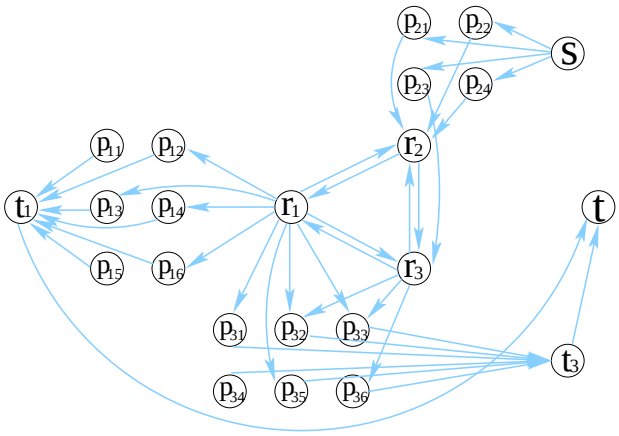


Figure 5.6 The corresponding flow network when b_2 is the source block.

PBO-Flow on G_i, the nets belong to $\bar{N}_i \cap \bar{N}_{i+1}$ should have already been found. Thus we only need to consider $\bar{N}_{i+1} - \bar{N}_i - ... - \bar{N}_1$ while pushing flows on G_{i+1}.

On the other hand, as we notice that a net belonging to $\bar{N}_i \cap \bar{N}_{i+1}$ should correspond to a path $(p_u, r'_1, ..., r'_k, p_v)$ ($p_u \in P_i$, $p_v \in P_{i+1}$, $r'_i \in R, i = 1, ..., k$) in G_i. Then in the modified flow network G_{i+1}, the path $(p_v, r'_k, ..., r'_1, p_u)$ must exist since the edge connections among buffer blocks are not changed. Thus we can remove all paths connected to nodes in P_{i+1} by increasing the capacities along all these paths, and applying PBO-Flow with nets $\bar{N}_{i+1}$. The optimality of PBO-Flow guarantees that the cost will not increase.

Furthermore, even if the algorithm does not return a feasible solution, (e.g, no feasible solution exists), the second way still outperforms the first one since the optimality of PBO-Flow guarantees that the new solution won't become worse, i.e., it can either connect more nets or reduce the cost.

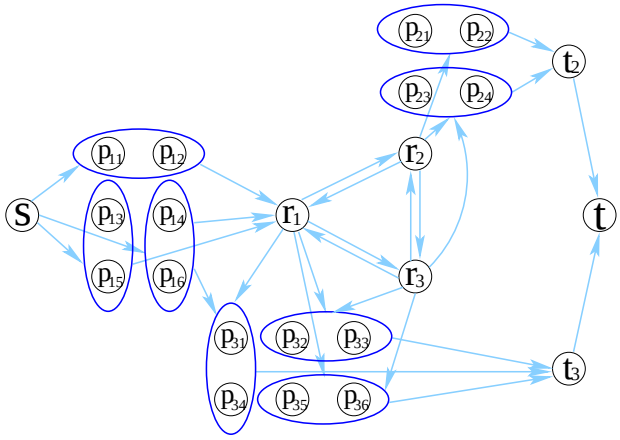


Figure 5.7 The corresponding flow network of the PB problem in Figure 5.5(b) using the node clustering method.

5.5 Improvement with Node Clustering

When we handle a big circuit which may include a huge amount of nets, the corresponding flow network might be quite large. In order to facilitate the process of huge PB problems, we propose the following node clustering method to speed up the computation.

For any $p \in P$, if p is inside a macro block, it must be connected to some pin outside the block. Without loss of generality, we may assume that a pin is only on the boundary of a macro block. Then by grouping neighbor pin nodes together, we can greatly reduce the number of nodes, consequently reducing the number of edges. Once several nodes are grouped together, we can use the average coordinate as the location of the new "supernode". And the capacity of the supernode is the number of nodes it includes. Figure 5.7 illustrates a constructed flow network for Figure 5.5(b) when 2 neighbor pins are clustered to one supernode. Actually, we can set different scale rates to blocks according to their sizes or

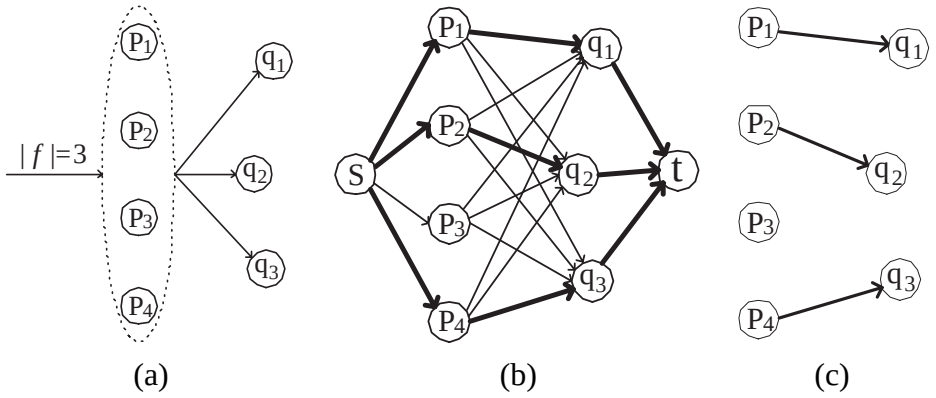


Figure 5.8 (a) A flow f, $|f| = 3$ flows through a supernode. (b) The corresponding network and a flow solution. (c) Deriving connections for original pin nodes.

other requirements.

Once we get a solution from the supernode flow network, we need to map the flow to a solution of the original PB problem, i.e, distributing the flows through one supernode to its pin nodes. Of course, we hope the mapping has the minimum connection length. In Figure 5.8(a), a flow f, $|f| = 3$ flows through a supernode which includes four pin nodes. 5.8(c) shows a feasible mapping. Still it can be solved with min-cost maximum flow as shown in Figure 5.8(b). Each node p_i $(i = 1, 2, 3, 4)$ in the super-node is connected to the destination nodes q_j $(j = 1, 2, 3)$ if $d_{p_i q_j}$ satisfies the distance constraints. The capacity of each edge is 1. The cost of edge (p_i, q_j) is the distance of two nodes $d_{p_i q_j}$. All other edges have a cost 0. Min-cost maximum flow guarantees to find an optimal mapping.

5.6 Experimental Results

Our algorithms were implemented in C++ on Sun Sparc Ultra 5 (360 MHz) with 128MB memory. We tested PB-Flow on 6 circuits which were generated randomly. For all files,

we adopted the node clustering method in Section 5.5 to reduce runtime.

We compared PB-Flow algorithm with a two-step approach (first assign pins, then plan buffers). We used a classical way [40] to assign pins as follows: by connecting the centers of two macro blocks, we got two crossing points on the boundaries of each block, then assign pins around the crossing points for the nets between these two macro blocks. After pin assignment was done for all nets, we used a net-by-net approach for buffer planning. The net-by-net approach considered only one net each time and found the min-cost path between the two pins of the net for buffer insertion.

For each test circuit, we repeated both approaches 5 times. Table 5.1 listed the average results of these five times. Using PB-Flow, we could find a feasible solution of pin assignment and buffer planning for all of the nets with a significant improvement on both the total wire length and the number of buffers. The last two rows show the comparison of the number of buffers and the total wire length. For both methods, we listed the test results and the values per net. The percentage was calculated according the values per net since the numbers of found nets are different. The last two testing files $F110$ and $M200$ include more than 4000 nets. The net-by-net approach can not handle such large files. For comparison purpose, we first apply the node clustering strategy to group pins, then use net-by-net for buffer insertion.

Table 5.1 Average results of PB-Flow for 5 times. All nets are found using PB-Flow algorithm.

File		A33n	X40	H80	S100	F110	M200
Grid		107x106	146x160	235x230	232x227	367x401	587x560
Blocks		33	40	60	91	110	100
Buffer Blocks		30	40	120	90	90	120
Nets		640	1021	1317	2021	4180	5659
Time (second)	Net-Net	27.80	34.49	75.46	119.67	91.91	135.77
	PB-Flow	41.91	27.32	67.94	83.12	220.45	326.9
Found nets	Net-Net	528.4	940.2	1316	1994	4167	5578
	PB-Flow	640	1021	1317	2021	4180	5659
Buffers	Net-Net (per net)	391.8 (0.741)	1731.8 (1.842)	2259 (1.717)	6400.2 (3.210)	9964.4 (2.391)	15463 (2.772)
	PB-Flow (per net)	327 (0.511)	1439 (1.409)	1949.8 (1.480)	5627 (2.784)	8631.4 (2.065)	13873.4 (2.452)
	reduced	31.04%	23.51%	13.80%	13.27%	13.63%	11.54%
Wire Length	Net-Net (per net)	920.2 (1.741)	2672 (2.842)	3575 (2.717)	8394.2 (4.210)	14131.4 (3.391)	21041 (3.772)
	PB-Flow (per net)	967 (1.511)	2460 (2.409)	3266.8 (2.480)	7648 (3.784)	12811.4 (3.065)	19532.4 (3.452)
	reduced	13.21%	15.24%	8.72%	10.12%	9.61%	8.48%

5.7 Conclusion

In this chapter, we presented a polynomial-time algorithm for simultaneous pin assignment and buffer planning for all two-pin nets between a source macro block and all other blocks such that each net satisfies the lower and upper bound of connection intervals as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$. By applying this algorithm iteratively (each time pick one block as the source block), it provides a polynomial-time algorithm for pin assignment and buffer planning for nets among multiple macro blocks. Experimental results demonstrate that the algorithm is very efficient and effective.

CHAPTER 6

ECO ALGORITHMS FOR REMOVING OVERLAPS BETWEEN POWER RAILS AND SIGNAL WIRES

6.1 Introduction

In ECO, a design needs to go through many changes [41, 42] due to constraints or target changes from manufacturing, marketing, reliability, or performance. At each step, designers usually want to modify the existing solution incrementally and keep the design as close as possible to the existing one. For high-volume high-revenue multiple-year design products, power rails may be changed due to added power rails for higher reliability, post silicon discovery, or design changes (such as cache size changes due to market reasons) that may not be predesigned in previous tapeout designs. And the post silicon debugging mandates the design to be fixed in previous converged design. This is different from ASIC or foundry model ECO where the original design assumption may change during manufacturing process according to its final volume recipe to tune yield and performance. Therefore, we cannot risk already highly invested design convergence efforts and schedule to redo ECO routing that changes the routing topology and needs to spend time again on converging the

design with no guarantee.

In this chapter, we address the PSO (Power rail - Signal wire Overlap) problem and propose two algorithms to solve it. PSO problems are usually caused by design changes in power delivery or package. For most high-performance VLSI design where mesh power rails are used to provide dense power supply for better signal integrity, the most upper metal layer can be changed due to different reasons such as current consumption requirement changes, package connection changes when the low resistance highest metal are used to deliver power, local routing ECO changes, etc. While the changes of power rails on the top metal layer may lead to the introduction of design rule violations between power rails and signal wires. It is not wise to rip up signal wires completely and reroute them for new power rails drop-in without keeping routing topology that has already proven converged in timing. Therefore, an efficient and graceful solution to PSO is very important due to design constraints and tight schedules during the late ECO stages. Furthermore, it is important to minimize disturbance upon the existing converged design while implementing ECO requests.

Informally, the PSO problem can be described as follows. On a multiple layer routing region with fixed power rails P on the top layer, there is a clean signal wire routing design which has no wire separation violations. Now a new design of power rails $\bar{P}$ on the top layer is adopted to replace the existing P. Some power rails in $\bar{P}$ may overlap with signal wires or the horizontal/vertical spacing between two wire segments is less than the wire separation requirement. Without loss of generality, we assume the multiple layer routing

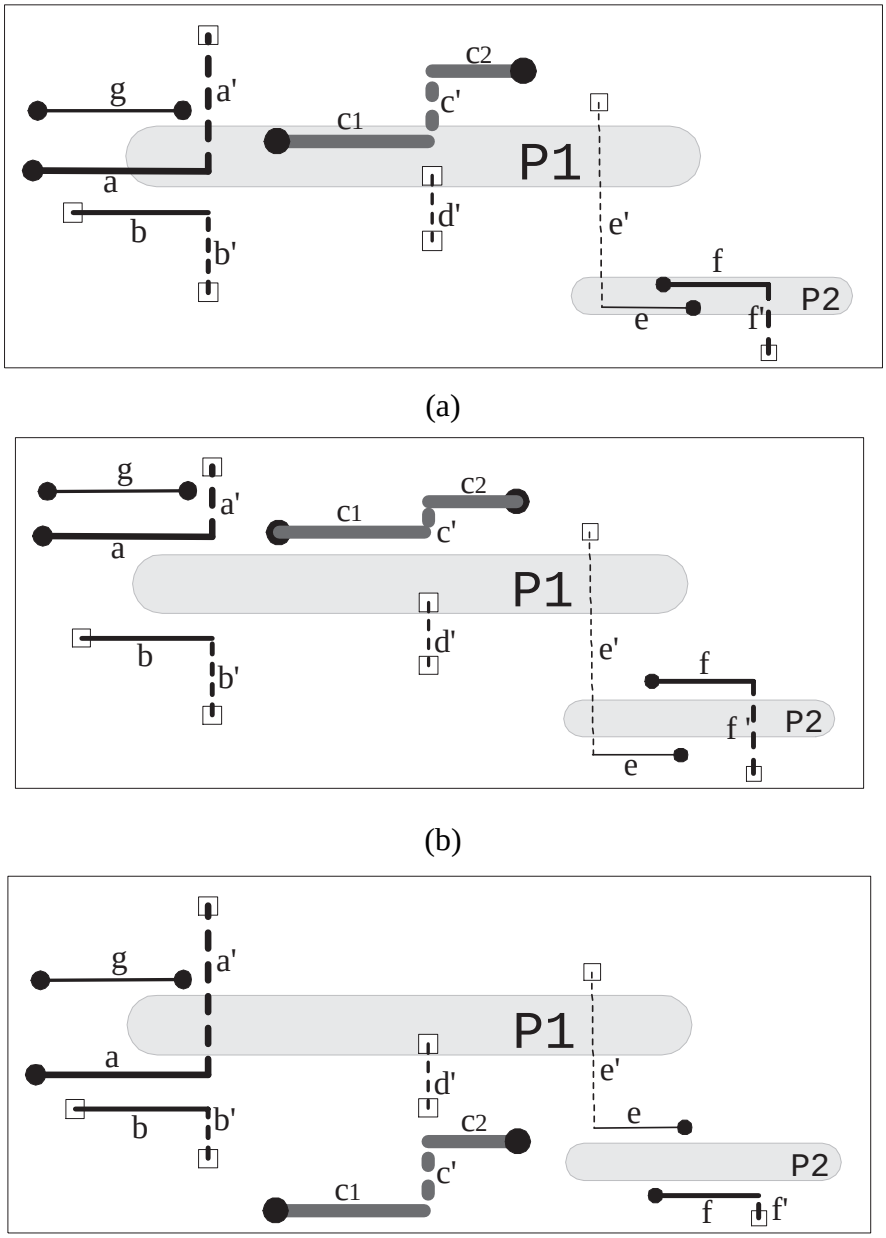


Figure 6.1 (a) Some horizontal signal wire segments on M_5 overlap with P_1 and P_2. (b) A feasible PSO solution. (c) A solution with violations.

design is in HVHVH style. We refer to the top layer as M_5 and the second top layer as M_4. M_5 is used for horizontal tracks, and M_4 is used for vertical tracks. Figure 6.1 shows an example. We use solid horizontal lines to present the top layer M_5 routing, and doted lines are for M_4 routing. Segments of different nets may have different widths. Due to the introduction of the new power rails P_1 and P_2 (the shadow area), some segments on M_5 overlap with P_1 and P_2.

Our purpose is to find a new clean routing solution without wire spacing violations. At the same time, we hope to keep the topology of the new routing solution as close as possible to that of the original one. Thus four constraints are set: (1) Keep the routing of power rails in the new design unchanged. This is required by the problem itself. (2) Only the routing of the top two layers is changed. The changes on the top layer M_5 may cause changes on M_4. However, the changes should not propagate to all layers. By treating all connections to M_4 from lower layers as fixed pins, the changes can be restricted to the top two layers. (3) Horizontal signal wire segments on the top layer M_5 can only move up/down. At the same time, keep the routing design pattern unchanged. This requires the following: (i) If one end point of a horizontal (vertical) wire segment on the top layer is a fixed pin, this segment cannot move. This kind of segments is called "P-segment". And its unmovable property is called "P-constraint". In Figure 6.1, fixed pins are denoted by a tiny square. But some pins on M_5, which are end points of horizontal segments, are allowed to move with the corresponding horizontal segments. They are unfixed pins and they are presented by solid dots. In Figure 6.1, horizontal segment b is a P-segment and cannot be moved like Figure

6.1(c). Fixed pins and unfixed pins come from signals passing through the local power rail ECO area where some of them are absolutely critical and are not allowed to be changed, while some of them have some minor freedom to shift. Furthermore, since pins on M_4 may connect to both M_5 and M_3, keep all pins on M_4 fixed so that there is no influence to lower layers. (ii) If vertical (horizontal) projections of two horizontal (vertical) signal wire segments have overlaps, then the up/down (left/right) relationship should not be changed. This is called "order consistency". For example, in Figure 6.1(c), horizontal segment e is above f while in the original design, e is below f. So this violates order consistency. (iii) If two horizontal (vertical) segments belonging to different nets are on the same track, their left/right (up/down) relationship should not be changed as long as the two segments still exist in the new solution. This is called "track consistency". In Figure 6.1(c), vertical segment c' is below d' while c' should be above d' to maintain track consistency. (4) For each signal segment, the deviation (i.e., the difference between its new position and the old one) should not exceed the user-defined allowable deviation bound so that local changes are confined. Different bounds can be set on different segments.

For the given new design of power rails $\bar{P}$, does there exist a clean routing solution satisfying the above constraints? If there is a solution, how to find one? For Figure 6.1, (b) gives a clean routing solution.

We notice that, once a segment is moved, it may overlap with some signal wire segments. Consequently these signal segments have to be moved, which may cause overlaps with other segments, etc. In Figure 6.1(b), the move-up of a causes g to move up too.

Furthermore, the movement of wire segments on the top layer may make some segments on M_4 become longer/shorter and introduce overlaps on M_4.

Note that our problem is significantly different from the overlap removal problem in macro-cell placement [43, 44, 45]. When macro cells are placed based on analytical techniques (e.g., force-directed placement, quadratic placement), there are overlaps among the macro cells. A clean-up phase is needed to shift the macro cells to remove all overlaps. However, PSO problem is significantly more difficult since moving the horizontal wire segments on M_5 also changes the vertical wire segments on M_4.

In this chapter, we first give the definition of PSO problem in Section 6.2. In Section 6.3, we give the definition of *FP-Range* and prove that if all segments move in their *FP-Range* as well as keeping order consistency and horizontal wire separation in the new routing solution, it satisfies the vertical wire separation requirement as well as track consistency and P-constraint. On the other hand, if a solution of a PSO problem exists, the new position of each segment must be in its FP-Range. In Section 6.4, we discuss the construction of the consistency graph. Based on FP-Range and consistency graph, we propose two polynomial-time algorithms PSO-H and PSO-G to solve PSO problems in Sections 6.5 and 6.6 respectively. Both algorithms guarantee to find a clean routing solution as long as one exists. PSO-H is faster than PSO-G, but PSO-G makes effort to minimize the total deviation. We show the experimental results in Section 6.7 and conclude the chapter in Section 6.8.

6.2 PSO (Power rail - Signal wire Overlap) Problem

A multiple layer routing region is three-dimensional. For convenience, we call the three dimensions the x, y and z dimensions. Each layer is an x-y dimensional plane, where x-axis goes horizontally and y-axis goes vertically. For convenience, let the coordinate of its bottom left corner be $(0, 0)$, and W and H be the width and the height of the routing region respectively. In most existing high-performance chips, each layer has its specific track orientation in either horizontal or vertical and its design rule which specifies minimum width and separation of wires. Let s be the half minimum wire separation of a metal layer. To simplify the presentation, suppose the wire separation requirement is $2s$ for both M_4 and M_5. The algorithms proposed in this chapter can be easily extended to handle different wire separations for different layers.

A horizontal segment can be presented by (x_1, x_2, y, w) where (x_1, y) and (x_2, y) are the end point coordinates of the center line ($x_1 < x_2$), and w is the half-width of the segment. Similarly, a vertical segment can be presented as (y_1, y_2, x, w). For any two horizontal segments (x_1, x_2, y, w) and (x'_1, x'_2, y', w') ($x_2 < x'_1$), if $(x_1-s, x_2+s) \cap (x'_1-s, x'_2+s) \neq \phi$, $|y-y'| \geq w+w'+2s$ must hold; also if $(y-w-s, y+w+s) \cap (y'-w'-s, y'+w'+s) \neq \phi$, $|x_2 - x'_1| \geq 2s$. The similar rule applies to vertical segments. Figure 6.2 gives an example of three horizontal segments. We say a routing solution is a clean solution if it has no overlap or wire separation violations. Sometimes we can simplify the presentation. For example, if we do not care the width of a horizontal segment, it is presented by (x_1, x_2, y).

Given a clean routing solution S with N signal nets, there are P power rails on the

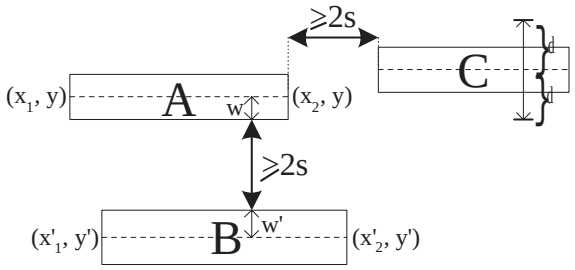


Figure 6.2 Wire separation requirement illustration.

top layer M_5. Also each signal segment is associated with a nonnegative number d called *allowable deviation bound*, i.e., for a horizontal segment (x_1, x_2, y, w), when it moves up/down, its new position $(x_1, x_2, \bar{y}, w)$ should satisfy that $|\bar{y} - y| \leq d$. Now if this design P is replaced by a new power rail design $\bar{P}$, the wire separation requirement might no longer be satisfied. How to modify the existing routing solution S so that the new routing solution $\bar{S}$ is a clean routing solution as well as satisfying the following constraints?

1. The power rails $\bar{P}$ on the top layer are not changed.

2. Only the routing of the top two layers M_4 and M_5 can be changed.

3. Horizontal signal wire segments on the top layer can only move up/down, i.e., the x-coordinate of the two end points of the segment keep unchanged. This is called "Shift movement".

4. The difference between the new position of a wire segment and its old location should not exceed its allowable deviation bound d.

5. If an end point of a signal wire segment on the top layer is a fixed pin, it is called

"P-segment" and cannot be moved. This property is called "P-constraint". All pins on lower layers are treated as fixed pins.

6. Suppose any two horizontal signal wire segments on M_5 (x_1, x_2, y) and (x'_1, x'_2, y') (assume $y > y'$) have new positions $(x_1, x_2, \bar{y})$ and $(x'_1, x'_2, \bar{y}')$, respectively, in the new routing solution $\bar{S}$. If $(x_1 - s, x_2 + s) \cap (x'_1 - s, x'_2 + s) \neq \phi$, $\bar{y} > \bar{y}'$ must hold. This property is called "order consistency".

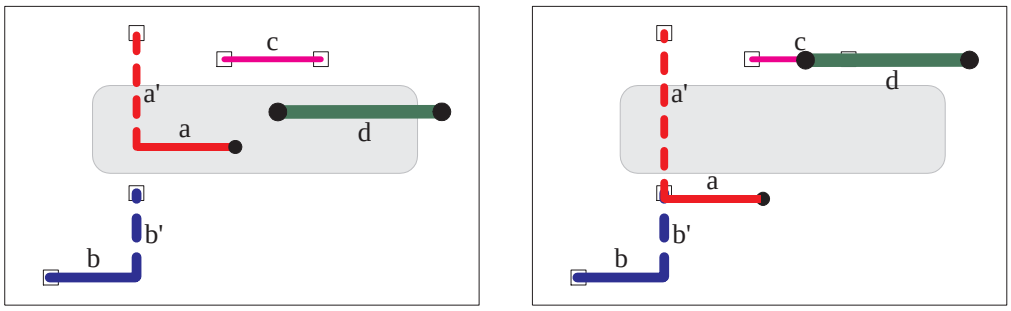
7. If two vertical signal wire segments (y_1, y_2, x, w) and (y'_1, y'_2, x', w') (assume $y_2 < y'_1$) belonging to different nets on the same layer satisfy $(x - w - s, x + w + s) \cap (x' - w' - s, x' + w' + s) \neq \phi$, then in the new design $\bar{S}$, if the two segments still exist, $y_2 < y'_1$ holds. This property is called "track consistency".

6.3 FP-Range (Fixed-Pin-decided Range)

In a PSO problem, since the original power rail design P is replaced by $\bar{P}$, we can just ignore P and simplify the problem as solving the overlaps between power rails and signal wire segments on M_5 as well as satisfying all of the constraints.

If we arbitrarily move one horizontal signal wire segment up or down, both horizontal overlaps between segments on M_5, and vertical overlaps on M_4 may be introduced. For example, in Figure 6.3(b), if horizontal segment d moves up, it overlaps with c. On the other hand, if a moves down, the vertical segments a' and b' on M_4 overlap with each other.

In this section, we discuss how to avoid vertical segment overlaps. The following theo-



(a) (b)

Figure 6.3 (a) A PSO problem. (b) Overlaps: vertical segments a' and b' on M_4; and horizontal segments c and d on M_5.

rem provides a rule of moving horizontal signal wire segments without introducing vertical wire separation violations. Also the theorem proves that if a horizontal signal wire segments moves out of its FP-Range, at lease one of the moving requirements, i.e., order consistency, track consistency, or P-constraint, or wire separation requirement, does not hold any more. To simplify the presentation, we neglect the widths of wire segments. It is easy to incorporate it into the following theorem. Each item is presented by its xy-coordinates, e.g., one segment is presented as (x_1, x_2, y) and one of its end point is denoted as (x_1, y).

Suppose the wire separation requirement is $2s$. A horizontal wire segment $R = (x_1, x_2, y_r)$ on M_5 belongs to net n_r. And its two end points are $r_1 = (x_1, y_r)$ and $r_2 = (x_2, y_r)$. If R is a P-segment, it cannot be moved. Its movable range is $[y_r, y_r]$. Otherwise, calculate two pin sets $\tilde{P}$ and $\tilde{Q}$. If r_1 is an unfixed pin, $\tilde{P} = \phi$; otherwise let $\tilde{P}$ be the set of fixed pins whose x-coordinate falls in $(x_1 - 2s, x_1 + 2s)$ and do not belong to net n_r. Also if r_2 is an unfixed pin, $\tilde{Q} = \phi$; otherwise let $\tilde{Q}$ be the set of fixed pins whose x-coordinate fall in $(x_2 - 2s, x_2 + 2s)$ and do not belong to net n_r. Let $U = min\{\{y - 2s | y \in \tilde{P} \cup \tilde{Q} \wedge y \geq y_r\} \bigcup \{H - 2s\}\}$ and $V = max\{\{y + 2s | y \in \tilde{P} \cup \tilde{Q} \wedge y \leq y_r\} \bigcup \{2s\}\}$. The range $[V, U]$ is

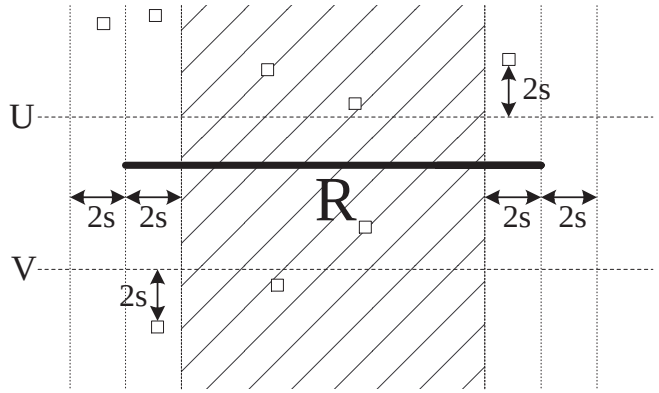


Figure 6.4 FP-Range illustration. The tiny squares are fixed pins.

called "FP-Range". Figure 6.4 shows the FP-Range of a horizontal segment R. As we notice that, if a vertical segment is created or becomes longer/shorter, it is caused by the end points of a horizontal segment. Therefore vertical overlap or vertical wire separation violation is only related to the end points of horizontal segments. When moving R up/down, the pins or segments in the shadow area cannot create vertical overlaps with R.

For any two end points a and b on layer M_4, suppose their coordinates are (x_a, y_a) and (x_b, y_b) respectively. And in a new routing solution, their positions are $(x_a, \bar{y}_a)$ and $(x_b, \bar{y}_b)$ respectively.

Lemma 6.1 *Let all horizontal segments on the top layer M_5 move up/down within their FP-Ranges $[V, U]$ and satisfy horizontal wire separation requirement and order consistency. For any two end points a and b on M_4, if $|x_a - x_b| \leq 2s$, then if $\bar{y}_a \geq \bar{y}_b$, $|\bar{y}_a - \bar{y}_b| \geq 2s$ and $y_a \geq y_b$.*

Proof

Each end point of a wire segment is a pin or it is connected to the other layer through a

via. Due to the shift movement of horizontal segments, some vertical segments are created and some disappear. If a vertical segment $T = (y_1, y_2, x_t)$ is created when a horizontal segment moves up/down, one end point of T must be a pin on M_4. Suppose the pin locates at (x_t, y_1). Since T does not exist in the original design, we let T's end points correspond to the end points of a empty vertical segment $E = (y_1, y_1, x_t)$, which connects to T through a via. In this way, for each end point in the new design, we can find its corresponding end point in the original one.

For convenience, if an end point is an end point of a horizontal segment or it is connected to a horizontal segment through a via, we call this kind of end points "turning point"; otherwise, it is called "fixed point". If an end point is a turning point in the new design, it must be a turning point in the original one since horizontal segments only move up/down. For a fixed point, there is no horizontal segments on M_5 connecting to it. So it must be a fixed pin on M_4 or connected to a fixed pin on M_5 through a via. Therefore, the location of a fixed point is not changed in the new design.

There are four cases:

1. Both a and b are fixed points. $y_a = \bar{y}_a$ and $y_b = \bar{y}_b$. Obviously, if $\bar{y}_a \geq \bar{y}_b$, $y_a \geq y_b$. Also in the original design, $|y_a - y_b| \geq 2s$, therefore, $|\bar{y}_a - \bar{y}_b| \geq 2s$.

2. Both a and b are turning points. They are connected to horizontal segments through vias. Suppose a and b correspond to horizontal segments A and B, respectively. Since $\bar{y}_a \geq \bar{y}_b$, A is above B in the new routing solution. Also the shift movements of A and B stick to "order consistency" as well as satisfying horizontal wire separation

requirement, so $|\bar{y}_a - \bar{y}_b| \geq 2s$, and A must be above B in the original routing solution. Thus $y_a \geq y_b$.

3. Point a is a fixed point and b is a turning point. b must be connected to a horizontal segment through a via. Let the horizontal segment be B. Since a is fixed, $\bar{y}_a = y_a$. Suppose $y_a < y_b$. According to the calculation of B's FP-Range $[V, U]$, $y_a < V$. From $V \leq \bar{y}_b$, $y_a < \bar{y}_b$, i.e., $\bar{y}_a < \bar{y}_b$. This contradicts with our assumption that $\bar{y}_a \geq \bar{y}_b$. Therefore, $y_a \geq y_b$. Since P_v is a fixed pin, and $y_a \geq y_b$, we know $y_a \geq U + 2s > U \geq \bar{y}_b$ according to the calculation of FP-Range. Therefore $|\bar{y}_a - \bar{y}_b| \geq 2s$.

4. Point a is a turning point and b is a fixed point. The proof is similar to case (3). ¶

Theorem 6.1 *If all horizontal segments on the top layer M_5 move up/down within their FP-Ranges $[V, U]$ and satisfy horizontal wire separation requirement and order consistency, the new routing solution has no vertical wire separation violation as well as satisfying track consistency and P-constraint. On the other hand, if one horizontal segment moves out of its FP-Range, order consistency, wire separation or track consistency or P-constraint violation is introduced.*

Proof

First, we prove that if all horizontal segments move within their FP-Ranges as well as maintaining order consistency and horizontal wire separation, no violation of P-constraint or track consistency or vertical wire separation is introduced. (If two segments have wire

separation violation, we also say they are overlapped.)

If a horizontal segment is a P-segment, its FP-Range makes it unmovable and causes no changes to the new routing solution. If both end points of a horizontal segment are floating pins, it can be moved up/down freely and causes no changes to M_4.

From Lemma 6.1, we know the relative positions of end points from different nets are not changed in the new design. Therefore, the track consistency is kept. Obviously, the P-constraint is satisfied. We only need to prove that if horizontal segments move within their FP-Range $[V, U]$ as well as keeping order consistency and horizontal wire separation, no vertical segment overlaps are introduced.

Suppose this claim does not hold. Then there exists a horizontal segment $A = (x_1, x_2, a)$ whose new position $(x_1, x_2, \bar{a})$ falls in its FP-Range $[V, U]$, and it causes overlaps between two vertical segments $(\bar{a}, \bar{b})$ and $(\bar{c}, \bar{d})$ on M_4 in the new routing solution. (We use y-coordinates to denote a vertical segment.) The two segments must belong to two different nets since overlap only occurs between two different nets. Suppose the two vertical segments in the original design is (a, b) and (c, d). They could be an empty segment. (For example, if no vertical segments are connected to A at (x_1, a) through a via, (x_1, a) is a pin on M_4. $(\bar{a}, \bar{b}, x_1)$ is a new segment and $\bar{b} = a$. Let it correspond to an empty segment (a, a, x_1) in the original design.) Without loss of generality, suppose $\bar{a} \leq \bar{c}$. Then their positions can be classified into three cases as shown in Figure 6.5.

1. $|\bar{b} - \bar{c}| < 2s$, this case does not hold according to Lemma 6.1.

2. $\bar{a} \leq \bar{c} \leq \bar{b} \leq \bar{d}$. According to Lemma 6.1, we can conclude that $a \leq c \leq b \leq d$.

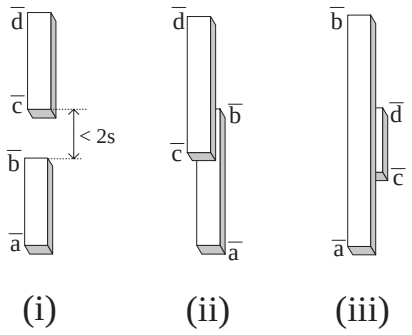


Figure 6.5 Three cases of vertical overlaps.

(i) $a \neq b$ and $c \neq d$. (a, b) and (c, d) are vertical segments in the original solution. But this contradicts the fact that the original one is a clean solution.

(ii) $a = b = c = d$. Both (a, b) and (c, d) are empty segments; suppose they are created from fixed pins P_a and P_c, respectively. Since P_a and P_c belong to different nets, P_a and P_c have to be on different layers. Suppose P_a is on M_5 and P_c is on M_4. Then P_a is an end point of a horizontal segment A. However, A is a P-segment and cannot be moved; therefore (a, b) cannot be created. $a = b = c = d$ does not hold.

(iii) $a = b = c < d$. (c, d) is a vertical segment in the original design, P_a must be on M_5 above (c, d). Therefore, A is a P-segment, and the edge (a, b) cannot be created. $a = b = c < d$ does not hold.

3. $\bar{a} \leq \bar{c} \leq \bar{d} \leq \bar{b}$. According to Lemma 6.1, we can conclude that $a \leq c \leq b$ and $a \leq d \leq b$. Still this case does not hold. The proof is similar to case 2.

Thus, we have proved that no vertical overlaps, track consistency and P-constraint are introduced in the new design.

On the other hand, if a horizontal segment moves out its FP-Range, the constraints, i.e., order consistency, track consistency, P-constraint, wire separation, do not hold any more.

Suppose one horizontal segment $A = (x_1, x_2, y_a)$ moves out its FP-Range $[V, U]$ and all of the constraints still hold. Without loss of generality, assume A moves below V to $(x_1, x_2, \bar{y}_a)$. If A is a P-segment, it cannot be moved. Thus A cannot be a P-segment. Also if both of A's end points are unfixed pins, V is $2s$ and A cannot go below V. Therefore, we assume at least one end point a of A is not a fixed pin, i.e., a connects to a fixed pin on M_4 through a via or a vertical segment B through a via. Suppose this end point is $a = (x_1, y_a)$ and its new position is $\bar{a} = (x_1, \bar{y}_a)$. Also suppose V is decided by a pin P_v below A and its x-coordinate falls in $(x_1 - 2s, x_1 + 2s)$. For convenience, P_v also denotes its y-coordinate. Furthermore, we may refer to a horizontal segment as its y-coordinate.

1. $P_v \leq \bar{A} < V$. If P_v is on M_5, it has wire separation violation with $\bar{A}$. Thus P_v is on M_4. However, if a connects to a fixed pin p on M_4 through a via, P_v has wire separation violation with p. If a connects to a vertical segment B through a via, P_v has wire separation violation with B. Therefore, $P_v \leq \bar{A} < V$ does not hold.

   In the following cases, we suppose $\bar{A} < P_v$.

2. Point a connects to a fixed pin p through a via. When A moves down, a new vertical segment B is created connecting a and $\bar{a}$. If P_v is on M_4 or it has connection to M_4 through a via, it overlaps with B. If P_v is on M_5 and have no connection to M_4, P_v must be an end point of a horizontal segment $\bar{C}$ in the new design and $\bar{C}$ is above the $\bar{A}$. So in the original design, C is above A. However, we know that A is above C

since P_v is a fixed pin below A. Therefore, this case does not hold.

3. Point a connects to a vertical segment B through a via.

   (i) P_v is on M_4.

   (a) P_v is connected to a horizontal segment C through a via, and C is below A in the original design. Let $\bar{C}$ be C's new position. To maintain order consistency, $\bar{C}$ is still below $\bar{A}$ in the new design. Then there must be a vertical segment D connecting $\bar{C}$ and P_v. $\bar{A}$ is between $\bar{C}$ and P_v. Therefore, $\bar{A}$ cannot have connections to M_4; i.e., B disappears in the new design and $\bar{a}$ must be a pin. Thus in the original design, B connects $\bar{a}$ and a through vias. But P_v is between a and $\bar{a}$. This leads to contradiction.

   (b) P_v has no connection to M_5. P_v is an end point of a vertical segment D. If B still exists, it has to be above D in order to keep track consistency and A cannot be below P_v. If B does not exist in the new design, it means B connects a and $\bar{a}$ in the original design. But P_v is between a and $\bar{a}$. This leads to contradiction.

   (ii) P_v is on M_5.

   (a) P_v is an end point of a horizontal segment C. C is a P-segment and cannot be moved. Therefore, A cannot go below P_v in order to maintain order consistency.

   (b) P_v is connected to a vertical segment D through a via. D must still exist in the new design. If B still exists, it has to be above D in order to keep track consistency and A cannot be below P_v. If B does not exist in the new design, it

means B connects a and $\bar{a}$ in the original design. But B and D have overlap in the original design. This leads to contradiction. ¶

From the above discussion, we can conclude that each horizontal segment must satisfy its FP-Range in the solution of a PSO problem. In other words, if we cannot find a solution that makes all horizontal segments in their FP-Range as well as keeping order consistency and horizontal wire separation, the PSO problem has no solution.

Since the locations of fixed pins are fixed, the FP-Range of each segment can be pre-calculated. And when searching for a solution of a PSO problem, we only need to consider order consistency, horizontal wire separation and FP-Range, and do not need to consider M_4 any more.

Suppose the number of fixed pins is N_p. To calculate the FP-Range of a segment, it takes $O(N_p)$. However, we can use a look-up table to speed up the searching instead of checking all of the fixed pins.

Given a routing region (W, H) (W and H could be quite huge), claim a two-dimensional array MAP[$\lceil W/R_w \rceil$, $\lceil H/R_h \rceil$], where R_w and R_h are two positive numbers set by users. The elements of MAP are a set of pins. A pin with location (p_x, p_y) is put in MAP[$\lceil p_x/R_w \rceil$, $\lceil p_y/R_h \rceil$]. For any horizontal segment (x_1, x_2, y), we only need to search MAP[$\lceil (x_1 - 2s)/R_w \rceil ... \lceil (x_1 + 2s)/R_w \rceil$; $0 ... \lceil H/R_h \rceil$] and MAP[$\lceil (x_2 - 2s)/R_w \rceil ... \lceil (x_2 + 2s)/R_w \rceil$; $0 ... \lceil H/R_h \rceil$]. In this way, the running time can be greatly reduced.

Also this theorem can take width into consideration. Suppose pin width is the same as segment width. As illustrated in Figure 6.6, we only need to consider the fixed pins falling

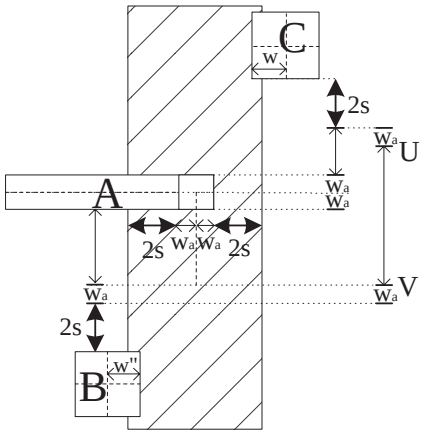


Figure 6.6 Illustration of the FP-Range calculation when the width is taken into consideration.

in the rectangular area. Also the $[V, U]$ bound should take the width of segment A into consideration, that is $[V + w_a, U - w_a]$.

Furthermore, this theorem can handle different spacing requirements of different layers without any extra work. Let s_5 and s_4 be the half minimum wire separation of M_5 and M_4 respectively, and usually the upper metal layer M_5 has larger spacing requirement than M_4. Since FP-Range is used to avoid vertical segment overlaps on M_4 under the assumption of no horizontal overlaps on M_5, if we use s_4 to do calculate the FP-Range, the theorem still holds.

6.4 Consistency Graph

From the theorem, we know that the influence of M_4 can be totally reflected in FP-Range. Thus we only need to deal with horizontal signal wire segments on the top layer.

An important property of PSO problem is to keep "order consistency". Given any two

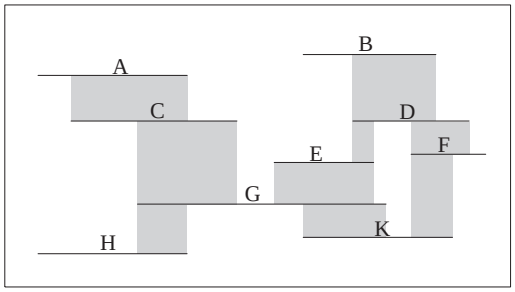
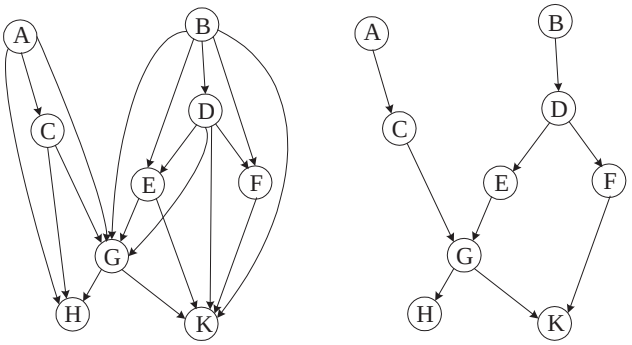


Figure 6.7 A routing solution of signal wires on the top layer



(a) (b)

Figure 6.8 (a) Full connections of adjacent segments. (b) Consistency graph.

horizontal segments $A = (x_{a1}, x_{a2}, y_a)$ and $B = (x_{b1}, x_{b2}, y_b)$, if $(x_{a1} - s, x_{a2} + s) \cap (x_{b1} - s, x_{b2} + s) \neq \phi$ ($2s$ is the wire separation requirement), we define segments A and B adjacent segments. According to order consistency, the relative positions of two adjacent segments should not be changed in the new routing solution. A good way to represent their relative positions is to construct a directed graph "consistency graph".

In the consistency graph, each node represents a horizontal signal wire segment. (Without misunderstanding, we use the same notation for segments and nodes, and refer to a node as its corresponding segment, vice versa.) For any two adjacent segments $\bar{A}$ and $\bar{B}$, if $\bar{A}$ is above $\bar{B}$, there must exist a path from $\bar{A}$ to $\bar{B}$.

One simple way to set up the consistency graph is to create an edge for each pair of

adjacent segments. Figure 6.7 gives a routing solution of signal wire segments on M_5. (Since we only consider horizontal segments, the picture only shows the top layer.) Figure 6.8(a) is its corresponding graph. But in this graph, a lot of edges are not necessary. For example, segment A is above C and C is above G, thus no necessary for edge (A, G).

Algorithm 8 Consistency-Graph-Construction (S_h)

```
1:  for each segment in S_h do
2:      create a node;
3:  end for
4:  for i = 1 to |S_h| do
5:      for j = i + 1 to |S_h| do
6:          if s_i and s_j are close adjacent segments then
7:              if s_i is above s_j then
8:                  add_edge(s_i, s_j)
9:              else
10:                 add_edge(s_j, s_i)
11:             end if
12:         end if
13:     end for
14: end for
```

For any two adjacent wire segments $A = (x_{a1}, x_{a2}, y_a)$ and $B = (x_{b1}, x_{b2}, y_b)$, let $[x_1, x_2] = [x_{a1} - s, x_{a2} + s] \cap [x_{b1} - s, x_{b2} + s]$ and ($y_a > y_b$). If there is no other segments overlapped with the rectangle (x_1, y_b, x_2, y_a) where (x_1, y_b) and (x_2, y_a) are the coordinates of the bottom-left corner and up-right corner, respectively, the two adjacent segments are called close adjacent segments, and one edge (A, B) is added. The rectangle (x_1, y_b, x_2, y_a) is called "clear box" if no other horizontal segments have overlaps with it. In Figure 6.7,

the shadow areas are clear boxes. Figure 6.8(b) shows a consistency graph by adding edges for each pair of close adjacent segments.

The construction of consistency graph can be summarized as *Algorithm 8* . S_h is the set of the horizontal signal wire segment on the top layer and a segment with index i is presented as s_i.

The consistency graph G is a planar graph and the number of nodes is $|S_h|$. So the number of edges is no more than $3|S_h|$. Then it takes $O(|S_h|)$ to check if two segments are close adjacent segments, and the graph construction involves two loops. Its worse case runtime is $O(|S_h|^3)$. However if we use a look-up table to record segments (similar to the pin look-up table in Section 6.3), the runtime can be greatly reduced.

6.5 PSO-H Algorithm

To solve the PSO problem, we draw on the consistency graph G to maintain the order consistency. For convenience, for any two nodes A and B in G, if there is a path from A to B, we say A is B's parent, and B is A's child.

Each time, select the nodes that have no parent nodes, and move them to their highest available positions. These positions are their new locations. Then remove these nodes from the graph. Repeat this process until no nodes are left.

For each segment, its available position is decided by its FP-Range, allowable deviation bound, the distribution of power rails, fixed pins on M_5, and the positions of its parents. Let the wire separation requirement be $2s$. Suppose segment $A = (x_1, x_2, y, w)$ has an

FP-Range $[V, U]$ and an allowable deviation bound d. If A moves in the range $[V, U] \cap [y - d, y + d]$, vertical wire separation, track consistency and P-constraint are satisfied according to Theorem 6.1. Obviously, the deviation bound is also satisfied. Also each node t records a value $Ubound$. $Ubound$ = $min\{y_r - 2s - w_r | y_r$ is the y-coordinate of a t's parent node and w_r is its width $\}$. Thus if A moves in the range $[0, Ubound - w]$, the order consistency and horizontal signal wire separation are guaranteed. Let $[\bar{V}, \bar{U}] = [V, U] \cap [y - d, y + d] \cap [0, Ubound - w]$. If no power rails fall in the rectangle region $(x_1 - 2s, \bar{U} - w - 2s, x_2 + 2s, \bar{U} + w + 2s)$, track $\bar{U}$ is the new position of the signal wire segment A. Otherwise, suppose a power rail segment $P = (x_{p1}, x_{p2}, y_p, w_p)$ has overlap with the rectangle region, let $\bar{U} = y_p - w_p - w - 2s$. Repeat the checking until a suitable position is found or $\bar{U} < \bar{V}$. The latter means no solution to the PSO problem. Furthermore, since power rails are checked segment by segment, the shorts of two power rail segments have no effect on the calculation. For fixed pins on M_5, they are handled similar to power rails.

PSO-H algorithm can be summarized as shows in *Algorithm 9*. S_h is the set of horizontal signal wire segments, P is the power rail set, R is the set of fixed pins, D records the allowable derivation bound for each segment. For each node t, its $Ubound$ is denoted as $t.Ubound$.

In this algorithm, each time we always put a horizontal segment to its highest available position. This leaves more room for other segments since once one segment is processed, its location is fixed and other segments below it cannot take the places above it. If we

arbitrary assign an available position, some segments may have no place to put.

Algorithm 9 PSO-H (S_h, P, R, D)

1: G = Consistency-Graph-Construction(S_h);
2: Calculate the FP-Range for each node;
3: Calculate [FP-Range] $\cap$ [allowable deviation];
4: Push all nodes without parent nodes into a List L;
5: **for** all nodes t in L **do**
6: $t.Ubound = H$;
7: **end for**
8: **while** $L \neq \phi$ **do**
9: Remove a node q from L;
10: Calculate q's new position;
11: **if** no position is found **then**
12: return "No Solution";
13: **end if**
14: Update q's child nodes' $Ubound$;
15: Delete q from G;
16: Push the nodes without parent nodes into L;
17: **end while**

In the calculation of available positions, FP-Range, deviation bound and the distribution of power rails are decided by the problem itself. If no range satisfies these three constraints, no solution. On the other hand, if the problem has a solution, the position of each signal wire segment in this solution must fall in the range of the available position obtained in PSO-H since PSO-H always puts segments to their highest available positions, therefore PSO-H guarantees to find a solution as long as one exists.

To construct the consistency graph, it takes $O(|S_h|^3)$. The calculation of FP-Range

may take $O(|R||S_h|)$. For the while loop, it has $|S_h|$ round. In each round, checking the intersection of power rails and fixed pins on M_5 may take $O(|P|+|R|)$. To update $Ubound$, each edge is visited only once for the whole loop. Since the number of edges is no more than $3|S_h|$, the runtime for PSO-H is $O(|S_h|^3+|S_h||R|+3|S_h||P|+3|S_h||R|)$. Still, we can set up loop-up tables for pins, signal wire segments and power rails so that the searching time can be greatly reduced.

Theorem 6.2 *Given a PSO problem, PSO-H algorithm guarantees to find a feasible solution in polynomial time as long as one solution exists.*

6.6 PSO-G Algorithm

In Section 6.5, we propose an algorithm PSO-H to solve PSO problem. In that approach, all horizontal signal wire segments are put to their highest available positions. Surely for some segments, this is not necessary. And in many applications, we hope to make the changes as little as possible. So in this section, we propose another algorithm PSO-G which tries to reduce the total deviation. The "total deviation" is defined as the sum of the deviations of all horizontal segments. Also we define the node overlapped with power rails as Onode; if an Onode overlaps a power rail $\hat{P}$, and it has no parent nodes overlapped with $\hat{P}$, the node is called URnode; Similarly, if an Onode has no child nodes in the same power rail, the node is called DRnode. And a Rnode refers to either a URnode or a DRnode.

Here are two observations:

1. Given a routing solution T, one solution T' is obtained by moving one Onode q

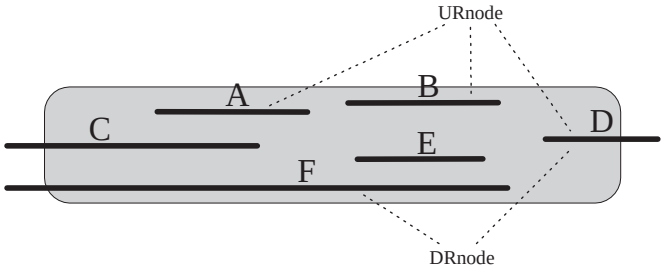


Figure 6.9 Illustration of Onodes/Rnodes.

outside power rail $\hat{P}$. To minimize the total deviation, the spacing between the new position of q and $\hat{P}$ must equal to the wire separation requirement.

2. Given a routing solution T, one solution T' is obtained by moving an Onode n_o which is not a Rnode outside power rails. Then there must be another solution T'' which is obtained by moving a Rnode outside power rails such that the total deviation of T'' is less than that of T'. This is because n_o has at least one parent and child in the same power rail. If n_o moves outside the power rail, it forces its parent/child nodes to move outside power rails in order to keep order consistency. Therefore, moving a n_o's parent/child node outside power rails leads to less total deviation. In Figure 6.9, the shadow area is a power rail. All segments inside are Onodes. A, B, and D have no parent nodes in this power rail. Thus A, B, and D are URnodes. F and D have no child nodes in this power rail. They are DRnodes. A node can be both URnode and DRnode, like D. Then the total deviation for moving the segment E must be larger than that for moving B or F.

Based on the two observations, we propose the PSO-G algorithm. In PSO-G, we need to search the graph in both directions (to parent nodes and to child nodes). To facilitate

searching, another set of edges is added. If there is an edge (a, b) in G, add an edge (b, a). To separate the two sets of edges, assign a color "black" to the original edges and "white" to newly added edges. If a node searches for its parents, use white edges; if it searches for child nodes, use black edges.

First, we assign a cost to each Rnode r. Suppose r is a URnode. If r is moved outside the power rail, the new position of r may overlap with other segments or it is above its parents' position. Then the affected signal segments are forced to move up accordingly. The cost of a Rnode r is the total deviation caused by moving r outside power rails. Since r is a URnode, the affected segments can only be its parents. Let the node r be the starting point. Using BFS(Breath-first search) algorithm on white edges, we can identify all possible affected segments and mark them "red". Assume r is on power rail $\hat{P}$, then its new position is $y_p + w_p + w_r + 2s$ where y_p is the y-coordinate of the center of $\hat{P}$, w_p and w_r are the width of $\hat{P}$ and r respectively, and $2s$ is the wire separation requirement. If this position is still overlapped with an power rail $\tilde{P}$, test $y_{\tilde{P}} + w_{\tilde{P}} + w_r + 2s$. Repeat this process until no overlaps with power rails or the position is outside r's available position. r's available position is the intersection of r's FP-Range and its allowable deviation range. If no suitable position is found, set its cost ∞. Otherwise, go ahead to process the affected segments. The position of an affected segment is not calculated until all its red child nodes have calculated their new positions. Suppose an affected segment is $A = (x_1, x_2, y, w)$. Let $z = max\{y_c + w_c + 2s | y_c$ is the y-coordinate of an A's red child node and w_c is its width $\}$. If z is not a suitable position, use the above procedure until an available position

is found. If no suitable position is found, the cost is ∞. If all affected segments can find a position, the cost is the sum of the difference of their new positions and the old ones. The similar rule applies if r is a DRnode.

Each time, select one Rnode with the minimum cost, move it outside power rails and adjust the positions of affected segments if the minimum cost is not ∞. Once a Rnode is moved outside power rails, it is just an ordinary segment and it is not a Rnode any more. Furthermore, some of its child/parent Onodes may become a Rnode. For each Rnode, recalculate the cost and repeat this process until there is no Rnode (i.e., no segments have overlap with power rails, and the result is a solution to the PSO problem), or until the minimum cost is ∞ (i.e., no solution to the PSO problem). The algorithm is summarized in *Algorithm* 10.

Algorithm 10 PSO-G (S_h, P, R, D)

```
1: Construct consistency graph, mark Rnodes;
2: Calculate the FP-Range for each node;
3: Calculate [FP-Range] ∩ [allowable deviation];
4: Push all Rnodes into List and assign costs;
5: while List ≠ φ do
6:     Select the Rnode r with the minimum cost;
7:     if min_cost = ∞ then
8:         return "No Solution";
9:     end if
10:    Change positions of r and affected segments;
11:    Push new Rnodes into List;
12:    Adjust/assign costs to Rnodes in the List;
13: end while
```

In *Algorithm* 10, the order consistency and horizontal wire separation are satisfied. Since all segments move within their FP-Range, no vertical wire separation or track consistency or P-constraint is violated according to theorem 1. Furthermore, each segment is in its deviation bound and has no overlap with power rails. Therefore, if PSO-G returns a solution, the solution is a correct one. On the other hand, if PSO-G cannot find a solution, the PSO problem has no solution.

If a URnode u has a cost ∞, at least two parent nodes p_1 and p_2 of u are overlapped or their up/down order is disturbed. Suppose p_1 is above p_2 in the original design. Since p_1 cannot move up further, it must have reached its highest possible position which is the intersection of FP-Range and allowable deviation bound excluding power rails. Also along the path from u to p_2, the spacing between two segments must be the required minimum spacing which is the sum of the widths of the two segments and $2s$. Thus if the cost of a URnode is ∞, it cannot be moved up outside power rails and this is totally decided by the problem itself instead of the processing method. Similarly, whether u can be moved down outside power rails is totally decided by the problem itself too.

If PSO-G returns "No Solution", the minimum cost must be ∞. For any Rnode r left in $List$, there are two cases: (1) The term r is a URnode and DRnode, r cannot be moved up or down outside power rails and it is decided by the problem itself. Therefore, no solution to PSO-problem. (2) Let r be a URnode and it has a child DRnode t. Both r and t have a cost of ∞ and cannot be moved down outside power rails. Still the PSO problem has no solution. Thus if PSO-G algorithm cannot find a solution, the PSO problem has no solution.

Suppose the number of segments overlapped with power rails is L. To construct the consistency graph and mark Rnodes, it takes $O(|S_h|^3 + |S_h||P|)$. The calculation of FP-Range may take $O(|S_h||R|)$. To calculate the cost of one segment, each edge is visited at most twice (First for BFS (Breath-First-Search), second for calculating the position) and it takes $O(|S_h||P|)$ since the edges of the consistency graph is at most $6|S_h|$ (including both black and white edges). Furthermore, each Rnode can be in the *List* at most twice. Thus the while loop has at most $2L$ rounds, and each round can be finished in $O(L|P||S_h|)$. Thus the total runtime is $O(|S_h|^3 + |S_h||R| + L^2|P||S_h|)$. By setting up look-up tables for pins, signal wire segments and power rails, the runtime can be greatly reduced.

Furthermore, once a URnode (DRnode) is moved outside power rails, the segments affected can only be its parents (children). So if the cost calculation of a Rnode does not involve these segments, its cost will not be changed after the URnode (DRnode) moves out, i.e., no cost adjustment (Line 12 in *Algorithm* 10) is needed for this node. Therefore, for each Rnode r, we record a bounding box $(x_{b1}, y_{b1}, x_{b2}, y_{b2})$ where (x_{b1}, y_{b1}) and (x_{b2}, y_{b2}) are the coordinates of bottom left corner and up right corner. The bounding box covers all of the affected segments when r moves out. After one URnode/DRnode t is moved out, if the bounding box of a Rnode has no overlap with r's bounding box, the cost keeps the same and no need for calculation. In this way, a lot of calculation can be saved.

Theorem 6.3 *Given a PSO problem, PSO-G algorithm guarantees to find a feasible solution in polynomial time as long as one solution exists.*

Table 6.1 Average results of PSO-H and PSO-G for 5 times.

File		N3	S6	M8	F10
ECO Region Area (um^2)		(4908.92, 3295.52)	(3295.52, 4908.92)	(10872.90, 4799.54)	(4799.54, 10872.90)
Top layer Signal Segments		1601	2098	1266	726
Power Rail Segments		166	1128	631	747
Overlapped Signal Segments		465	594	441	206
Allowable Deviation		2%	2%	2%	2%
Time (second)	PSO-H	5.62	9.17	3.81	3.26
	PSO-G	21.44	41.28	23.72	11.38
Max Deviation	PSO-H	1.987%	1.992%	1.991%	1.996%
	PSO-G	1.617%	0.231%	1.522%	0.944%
Total Deviation	PSO-H(um)	63029.58	126855.37	108874.92	120950.08
	PSO-G(um)	13097.40	1208.10	7066.87	4200.80
	PSO-G/PSO-H	20.780%	0.952%	6.491%	3.473%

6.7 Experimental Results

Our algorithms were implemented in C++ on PC (733 MHz) with 128MB memory. We tested PSO-H and PSO-G algorithms on four test files. These circuits were derived from industry files and the top layer is for horizontal tracks. Both approaches were repeated 5 times. Table 6.1 lists the average results of these five trials. For all of the test circuits, we can find a clean routing solution and the derivation of each signal segment is bounded as 2% of the height of the ECO region area. For PSO-H algorithm, each top layer signal wire segment is moved only once. So it is much faster than PSO-G algorithm. If the requirement only wants a clean solution satisfying the deviation bound, PSO-H is preferred. But if no

deviation bound is given or want to find a solution as close as possible to the original design, PSO-G algorithm always returns a solution with less total deviation and a smaller max-deviation.

6.8 Conclusion

In this chapter, we have presented two polynomial-time algorithms to solve the overlaps between power rails and signal wires on the top layer as well as satisfying the allowable deviation bound, order consistency, track consistency, P-constraint and wire separation requirement. Both algorithms guarantee to find a feasible solution as long as one exists. One is faster, while the other makes effort to reduce total deviation. According to different application requirements, users can choose an appropriate one. Experimental results demonstrate their efficiency and effectiveness.

CHAPTER 7

AN ECO ALGORITHM FOR ELIMINATING CROSSTALK VIOLATIONS

7.1 Introduction

Any changes on existing routing design may cause design rule violations and it is necessary to develop efficient and graceful algorithms to resolve these violations. In this chapter, we propose an algorithm (CVE) to eliminate crosstalk violations to a given routing design. The target is to find a new clean routing solution with no crosstalk violations under the constraints similar to the constraints stated in the previous chapter. Therefore, the CVE algorithm can also be applied to the output of the PSO problem.

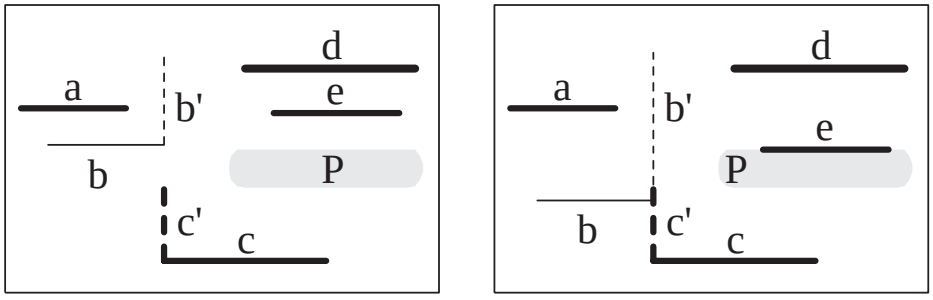


Figure 7.1 (a) A routing solution with crosstalk violations. (b) A routing solution with overlap violations.

As we notice, once a signal wire segment is moved, the total capacitive crosstalk on both this segment and its neighbor segments may be changed. At the same time, design spacing rule violations must be avoided. For convenience, if the spacing between two segments is less than the minimum spacing requirement, we say the two segments overlap. Figure 7.1(a) gives a routing solution with five horizontal signal wire segments, two vertical wire segments and one power rail. Suppose segments b and e violate the capacitive crosstalk requirement; i.e., the total crosstalk on b and e exceeds defined thresholds. As illustrated in Figure 7.1(b), if e is moved down, it overlaps with the power rail P. Also if b is moved down, vertical overlap between b' and c' on $\hat{L}$ is introduced.

In this chapter, we propose a two stage CVE (Crosstalk Violation Elimination) algorithm to eliminate crosstalk violations for a given routing design as well as minimizing the total deviation. The first stage FCVE processes signal wire segments on L one by one and tries to find a clean routing solution satisfying all constraints. Then in the second stage SCVE, we make efforts to minimize the total deviation based on the shortest path algorithm. Experimental results demonstrate that our approach is efficient and effective.

The rest of the chapter is organized as follows: Section 7.2 gives the definition the CVE problem, and Section 7.3 introduces some preliminaries. Then we propose a two-stage algorithm in Section 7.4. In Section 7.5, some optimization strategies are presented and experimental results are given in Section 7.6. We conclude the chapter in Section 7.7.

7.2 Crosstalk Violation Elimination

Given a routing solution S with N signal nets, there are P power rails on layer L. For convenience, let the coordinate of the left bottom corner of the routing region be $(0, 0)$. Suppose s is the half minimum wire separation of a metal layer. For a horizontal segment, it can be represented by (x_1, x_2, y, w, c, d) where (x_1, y) and (x_2, y) are the end point coordinates of the center line $(x_1 < x_2)$, and w is the half-width of the segment, c is called crosstalk threshold, i.e., the total capacitive crosstalk to the segment should not exceed this bound, and d is allowable deviation bound, i.e., when the segment moves up/down, its new position $(x_1, x_2, \bar{y}, w, c, d)$ should satisfy $|\bar{y} - y| \leq d$. Similarly, a vertical segment can be represented as (y_1, y_2, x, w, c, d). Sometimes, we can simplify the representation. For example, a horizontal segment can be represented by (x_1, x_2, y) if we do not care other factors.

Since the crosstalk on some sensitive segments in S exceeds the given bounds, the target is to modify the existing routing solution S so that the new routing solution $\bar{S}$ is a clean routing solution which satisfies the following constraints:

1. The power rails P on L are not changed.

2. Horizontal signal wire segments on L can only move up/down, i.e., the x-coordinates of the two end points of the segment keep unchanged.

3. The total crosstalk on a wire segment should not exceed its capacitive crosstalk threshold c.

4. The relative positions of two segments on all layers should not be changed.

   For example, for any two horizontal signal wire segments on one layer $(x_1,\ x_2, y)$ and (x'_1, x'_2, y') (assume $y > y'$), their new positions are $(x_1, x_2, \bar{y})$ and $(x'_1, x'_2, \bar{y}')$ in the new routing solution $\bar{S}$ respectively. If $(x_1 - s, x_2 + s) \cap (x'_1 - s, x'_2 + s) \neq \phi$, $\bar{y} > \bar{y}'$ must hold. Similar requirements for vertical segments. This property is called "order consistency".

5. The difference between the new position of a wire segment and its old location should not exceed its allowable deviation bound d.

   The term d is defined to constrain that one segment does not derive too much from its original position. At the same time, it helps to prevent introducing new crosstalk violations to other layers. When horizontal segments on L are changed, the length of vertical segments on $\hat{L}$ or $\tilde{L}$ may also be changed. However, the length change is no more than $2d$ since each vertical segment connects to at most two horizontal segments on L. Then the crosstalk introduced by length increase is also limited. Therefore, by setting appropriate deviation bounds, new crosstalk violations on layer $\hat{L}$ or $\tilde{L}$ can be avoided.

Although the CVE problem deals with crosstalk violations on one layer, it can be applied layer by layer to resolve violations on all layers to a given multiple layer routing design.

7.3 Preliminaries

7.3.1 FP-Range

Although the constraints of the CVE problem is slightly different from those of the PSO problem, we have the following theorem related to FP-Range.

Theorem 7.1 *If all horizontal segments on layer L move up/down within their FP-Ranges $[V, U]$ and satisfy horizontal wire separation requirement and order consistency, the new routing solution has no vertical wire separation violations.*

7.3.2 Crosstalk model

In general, each segment has coupling effect to all other segments. However, the coupling capacitance decreases dynamically if the segment is out of the neighborhood of the other segments [46, 47]. Therefore, we only consider the capacitive crosstalk between two neighboring parallel wires and suppose the neighborhood distance is $D = \gamma \cdot 2s$, ($0 < \gamma < 2$). Then the capacitive crosstalk between two segments can be expressed by the following formula:

$$c = \begin{cases} \alpha \cdot \frac{l}{t^2} & t \leq D \\ 0 & t > D \end{cases}$$

where α is the coupling parameter, l is the coupling length, and t is the distance between two segments.

In Figure 7.2, there are three wire segments A, B and C. Segments A and C have

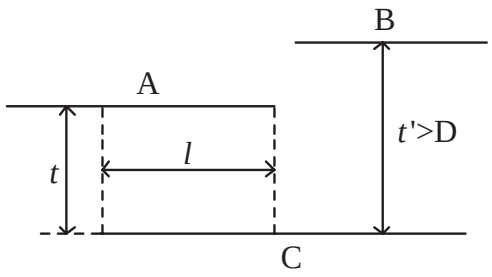


Figure 7.2 Segments A and C have capacitive crosstalk; while the crosstalk between segments B and C is zero.

capacitive crosstalk. While the crosstalk between segments B and C is zero since the distance between the two segments is larger than D.

Furthermore, for power rails, they act as a shield and do not cause crosstalk to their adjacent segments.

7.4 CVE Algorithm

To solve the CVE problem, we develop a two-stage algorithm. The first stage FCVE processes signal wire segments on L one by one and tries to find a clean routing solution satisfying all constraints. Then in the second stage SCVE, efforts are made to minimize the total deviation based on the shortest path algorithm.

7.4.1 FCVE algorithm

For convenience, for any two nodes A and B in G, if there is a path from A to B, we say A is B's parent, and B is A's child.

The main idea of FCVE algorithm is as follows: each time, select the nodes that have

no parent nodes and try to move them to their highest available positions. These positions are their new locations. Then remove these nodes from the graph. Repeat this process until no nodes are left.

For each segment, its available position is related to its FP-range, allowable deviation bound, crosstalk threshold, the distribution of power rails and the positions of its parents. Let the wire separation requirement be $2s$. Suppose segment $A = (x_1, x_2, y, w, c, d)$ has an FP-range $[V, U]$. Also A records a value $Ubound$. $Ubound$ = $min\{y_p - 2s - w_p | y_p$ is the y-coordinate of an A's parent node and w_p is its half width$\}$. Then if A moves in the range $[0, Ubound - w]$, the order consistency is guaranteed. Let $[\bar{V}, \bar{U}] = [V, U] \cap [y - d, y + d] \cap [0, Ubound - w]$. Check tracks t starting from $\bar{U}$. If t is not occupied by any power rails and no crosstalk violations are introduced to A's parents and itself if A is put at track t, t is assigned as A's new position. Otherwise, check the next track below t. Repeat this process until a feasible position is found or the track goes beyond $\bar{V}$. The latter case means no feasible solution is found. Once the position of A is decided, the capacitive crosstalk bounds of A and A's parents have to be adjusted accordingly, i.e., minus the crosstalk between A and its parent from the crosstalk bounds of A and its parent.

Furthermore, if one segment has several children, then the children selected first always have higher priority. For example, in Figure 7.3, suppose the position of A has been fixed. B, C and D are three children of A. The coupling length ratio of B, C and D is $2 : 1 : 1$. The crosstalk bound of all segments is 30. The numbers in the figure indicate the capacitive crosstalk if the segment is placed at their highest available positions. Suppose B is first

selected and it is placed as Figure 7.3(b). Then the crosstalk bound of A is reduced to 0. Therefore C and D have to be placed lower, which pushes E down too. In order to avoid one segment consuming all or most of the crosstalk budget, we use the following approach. Suppose a segment R is fixed and its crosstalk bound is c_r. Also its total coupling length with all of its unfixed children is l_r. Let $d_r = \min\{D, \sqrt{\alpha \cdot (l_r/c_r)}\}$. Then the distance between R and its first selected child T must be no less than d_r. Once T is fixed, c_r is adjusted accordingly, i.e., minus the crosstalk between R and T from c_r. Then the new c_r is used for R's other children in the same way. Figure 7.3(c) shows a solution with this approach. According to the crosstalk budget, the crosstalk between A and B, A and C, A and D should be 15, 7.5, and 7.5, respectively. Suppose segment B is first selected, then the crosstalk upper bound of A is reduced to 15. Since the lengths of C and D are the same, C and D get a crosstalk budget 7.5. And the new position of C can be calculated. However, the highest available position of C is lower than the calculated position. Therefore, C is put on its highest available position and the crosstalk to A is 7. Finally, D takes all of the crosstalk budget.

In FCVE, we always try to put a horizontal segment upwards. This leaves more room for other segments since once one segment is processed, its location is fixed and other segments below it cannot take the places above it. If we arbitrarily assign a segment to one of its available positions, some segments may have no place to put.

We notice that, even if there are no crosstalk violations in the given input routing design, segments may still be moved in the above procedure. However, our targets are not only to

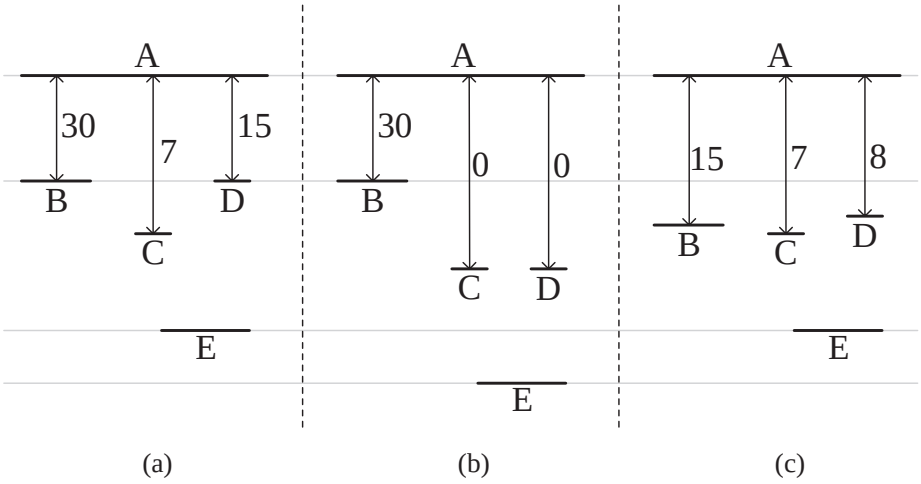


Figure 7.3 (a) B, C, and D are three children of A. The position of A is fixed. (b) B is first selected and put to its highest available position. (c) A solution according to our approach.

eliminate crosstalk violations, but also to minimize the total deviation. Therefore, we start with a zero allowable deviation bound, and each time we increase the bound by a certain percentage. For each deviation value, we calculate the positions of all segments according to the above procedure. We repeat this process until a feasible solution is found or the deviation bound exceeds the predefined value. For the latter case, no feasible solution is found.

The algorithm can be summarized as in *Algorithm 11*. S_h is the set of horizontal signal wire segments, P is the power rail set, F_p is the set of fixed pins. For each node v, $v.d$ is its allowable deviation bound and $v.Ubound$ refers to its $Ubound$. The term δ is the increase percentage for each iteration. Then $v.d * increase$ is the allowable deviation of node v. Suppose the width and height of the chip are W and H, respectively.

Algorithm 11 FCVE (S_h, P, F_p, δ)

1: G = Consistency-Graph-Construction (S_h);
2: Calculate FP-Range for each node;
3: $increase = 0$;
4: **while** $increase \leq 1$ **do**
5: Calculate [FP-Range] $\cap$ $[v.d * increase]$ of nodes v;
6: Push nodes without parent nodes into a List T;
7: **for** all nodes t in T **do**
8: $t.Ubound$ = H;
9: **end for**
10: **while** $T \neq \phi$ **do**
11: Remove a node q from T;
12: Calculate q's new position;
13: **if** no position is found **then**
14: $increase = increase + \delta$;
15: Restore G;
16: Goto 4;
17: **end if**
18: Update $Ubound$ of q's children;
19: Update crosstalk bounds of q and its parents;
20: Delete q from G;
21: Push the nodes without parent nodes into T;
22: **end while**
23: Return "New Solution";
24: **end while**
25: Return "No Solution"

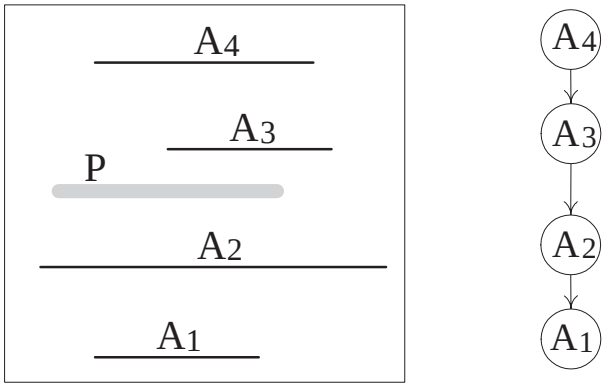


Figure 7.4 (a) A CVEP problem. There are 4 signal wire segments A_1, A_2, A_3, and A_4, and 1 power rail P. (b) The consistency graph is a path.

7.4.2 SCVE

If FCVE returns a solution, then the solution must be a feasible solution satisfying all of the constraints. However, FCVE tends to place segments to their "highest" available positions while some segments do not need to deviate so much from their original positions. In this section, we first consider a special case of CVE problem (CVEP) and propose an exact polynomial-time algorithm to decide wire segment positions with minimum total deviation under all constraints. Then by applying this algorithm repeatedly on the output of FCVE, we can greatly reduce the total deviation.

Problem 7.1 CVEP *is a special case of CVE problem when all horizontal segments on layer L are placed in a line, i.e., the corresponding consistency graph is a path.*

Figure 7.4(a) shows an example. There are 4 signal wire segments and 1 power rail. Figure 7.4(b) is its consistency graph and it is a single path from node A_4 to A_1. For convenience, segments in a CVEP problem are indexed as A_1, ..., A_n from bottom to top.

To solve the CVEP problem, we first construct a "Segment Position" (SP) graph, and then apply the shortest path algorithm to get the solution. The SP graph is constructed in two steps. The first step graph (FSP) $G = (V, E)$ is formed as follows.

1. Nodes: Since the allowable deviation of segment A_i is d_i, totally there are $2d_i + 1$ possible positions for A_i. Let node set $V' = \{v_i^j | i \in [1, n], j \in [-d_i, d_i]\}$ representing possible positions of A_i; i.e., v_i^j refers to the position $y_i + j$. For convenience, we call v_i^j a node of A_i. Also for any possible position, if it is occupied by a power rail or it is outside A_i's FP-Range, then A_i cannot put there. Suppose nodes corresponding to this kind of positions form the set V''. $V = V' - V''$.

2. Edge: $E = \{(v_i^j, v_{i+1}^k) |\ v_{i+1}^k - w_{i+1} - (v_i^j + w_i) \geq 2s, i \in [1, n-1], j \in [-d_i, d_i], k \in [-d_{i+1}, d_{i+1}],\ v_i^j \in V, v_{i+1}^k \in V\}$. For each node of A_i, it is connected to the nodes of A_{i+1} such that the distance between two nodes satisfies the minimum spacing requirement.

3. Cost: each edge (v_i^j, v_{i+1}^k) is assigned a cost which is the capacitive crosstalk between A_i and A_{i+1} supposing the two segments are placed at v_i^j and v_{i+1}^k respectively.

Figure 7.5 shows an example. (a) is a CVEP problem with 3 signal wire segments A_1, A_2, A_3 and 3 power rails. For simplicity, suppose all wires have the same length, and the deviation bounds of signal wire segments are all 2. Also the crosstalk thresholds are all 0, i.e., the distance between any two signal wire segments must be larger than 1 unit. In Figure 7.5(a), since segments A_1 and A_2 are adjacent to each other, the capacitive crosstalk

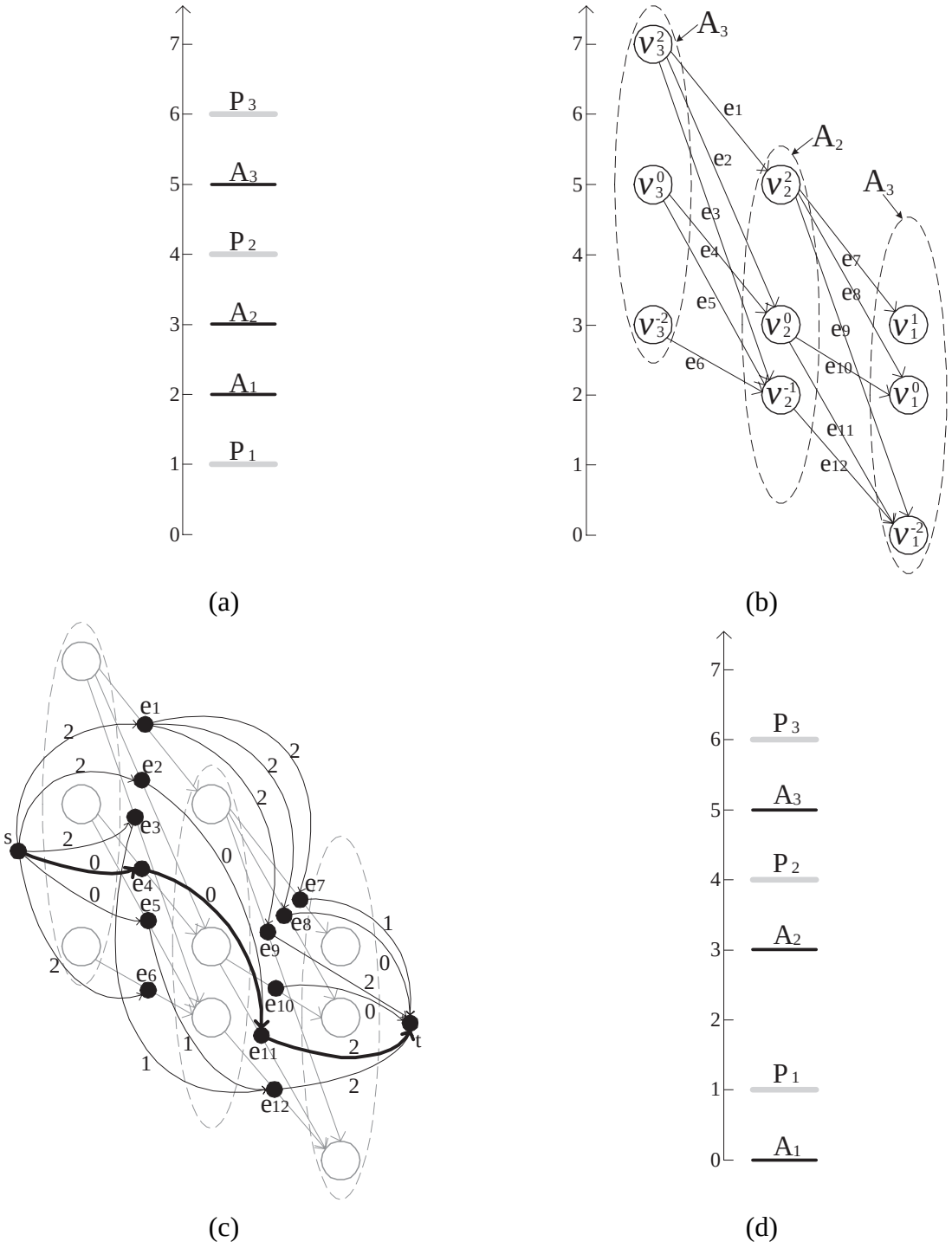


Figure 7.5 (a) A CVEP problem. (b) FSP graph G of the CVEP problem. (c) SP graph $\bar{G}$ of the CVEP problem. (d) A feasible solution to the CVEP problem.

between them exceeds the crosstalk bounds of both A_1 and A_2. Suppose A_1 and A_2 violate the crosstalk requirement.

Figure 7.5(b) shows the corresponding CVEP graph G for (a). Due to the overlap with power rails, the available positions of each segment are only 3 and they are represented by 3 nodes respectively. The costs of all edges are 0 except two edges e_6 and e_{10}.

In FSP graph, the allowable deviation bound is reflected by nodes, and the edge cost records the crosstalk between two segments. However, the crosstalk constraint is not included. Therefore, based on FSP graph, we derive the SP graph $\bar{G} = (\bar{V}, \bar{E})$ so that the shortest path algorithm can be applied to find the solution. $\bar{G}$ is formed as follows.

1. Nodes: Each edge in G is represented by a node. For convenience, a edge (u, v) in FSP graph also refers to a node in SP graph. Also two nodes s and t are added representing the starting and ending nodes respectively.

2. Edges: For any two edges (v_i^j, v_{i+1}^k) and (v_{i+1}^k, v_{i+2}^l) in FSP graph, if the total cost of the two edges is less than c_{i+1}, which is the crosstalk bound of segment A_{i+1}, an edge is added between the two corresponding nodes in $\bar{G}$. Also connect s to all of the nodes corresponding to the edges related A_1 in FSP graph, and all of the nodes corresponding to the edges related to A_n are connected to t.

3. Cost: If edge $\bar{e}$ connects two nodes (v_i^j, v_{i+1}^k) and (v_{i+1}^k, v_{i+2}^l), the cost of $\bar{e}$ is $|k|$ (i.e., the deviation of A_{i+1}); if edge $\bar{e}$ starts from s (i.e., $\bar{e}$ connects s and (v_1^j, v_2^k)), the cost is $|j|$; if edge $\bar{e}$ ends at t (i.e., $\bar{e}$ connects (v_{n-1}^j, v_n^k) and t), the cost is $|k|$.

Figure 7.5(c) illustrates the SP graph $\bar{G}$ for the given CVEP problem. Each edge in G is represented by a node in $\bar{G}$. For edges e_6 and e_{10} in G, since their cost is 1, edges (e_6, e_{12}), (e_2, e_{10}), and (e_4, e_{10}) are not included in $\bar{G}$. Based on $\bar{G}$, we apply the shortest path algorithm to find the shortest path from s to t. In Figure 7.5(c), the shortest path is indicated by thick curves. It is easy to derive a CVEP solution from the shortest path in $\bar{G}$ as shown in Figure 7.5(d).

Suppose totally there are n wire segments and M is the max allowable deviation. The number of nodes in FSP graph is $O(n \cdot M)$. For each node in FSP graph, it connects to at most M nodes. Therefore, the number of nodes and edges in SP graph $\bar{G}$ are $O(n \cdot M^2)$ and $O(n \cdot M^3)$, respectively. Since $\bar{G}$ is a directed acylic graph, the shortest path algorithm can be accomplished in $O(|\bar{V}| + |\bar{E}|)$ [14, 15], i.e., $O(n \cdot M^3)$.

We now summarize the CVEP algorithm as *Algorithm 12*.

Algorithm 12 CVEP (P)

1: Construct SP graph $\bar{G}$ for the input path P;
2: Apply shortest path algorithm on $\bar{G}$;
3: Derive the solution to the given CVEP problem

The construction of SP graph $\bar{G}$ takes $O(n \cdot M^3)$, and the derivation from a shortest path in $\bar{G}$ to a CVEP solution takes $O(n)$. Therefore, CVEP algorithm can solve CVEP problems in $O(n \cdot M^3)$. Furthermore, the algorithm guarantees to return a feasible solution with minimum deviation as long as there is a solution to the given CVEP problem.

Based on CVEP algorithm, we have the SCVE algorithm as *Algorithm 13*. SCVE algorithm performs as the second stage of CVE algorithm since its input is the output of

FCVE algorithm, which is a feasible solution to the given CVE problem. The target of SCVE is to reduce the total deviation.

Based on the consistency graph, each time we select a path and apply CVEP algorithm to find the optimal solution corresponding to the selected path. Once a path is processed, all nodes along the path are marked "Processed", and their positions are no longer changed. Since FCVE algorithm traverses a consistency graph from top to bottom, and many segments may put on a position higher than their original positions, SCVE algorithm selects paths from the bottom of a consistency graph. For each path, the first node u must either have no child or all of its children are marked. Then trace up to its parents. If one of its parents p has u as the only unmarked child, then p is selected and continue this procedure until no nodes satisfy the selection rule. Once a path is selected, we treat all other nodes unchanged and apply the CVEP algorithm. Note that the capacitive crosstalk of each possible position of a signal wire segment is also affected by other segments which are not incident on the path.

Algorithm 13 SCVE ()

1: Set all nodes in the consistency graph "UnProcessed"
2: **while** $\exists$ "UnProcessed" nodes **do**
3: Select a path P from the consistency graph;
4: Apply CVEP algorithm on P;
5: Mark all nodes on P as "Processed";
6: **end while**

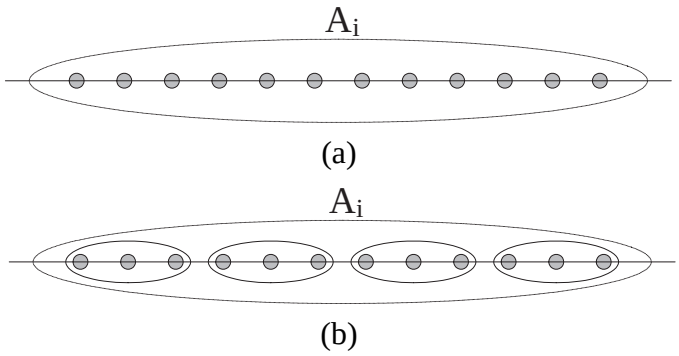


Figure 7.6 (a) A_i is a wire segment and it has 12 available positions. (b) Every three nodes are clustered as a "supernode".

7.5 Optimization

CVEP algorithm is the kernel part of SCVE algorithm. However, the number of possible positions of wire segments may be quite large and it makes SP graph include a lot of nodes and edges, which requires not only much memory but also long running time. In order to speed up the execution, we develop the following optimization strategies.

7.5.1 Node clustering

When the deviation of a wire segment is large, the corresponding FSP graph and SP graph must include a large number of nodes. In order to facilitate the process of huge CVEP problems, we propose the following node clustering method to speed up the computation.

For any wire segment A_i, suppose the number of its possible positions is M. Then by grouping neighbor positions together, we can greatly reduce the number of nodes in FSP graph, consequently reduce the size of SP graph. Once several nodes are grouped together, we can use the average coordinate as the location of the new "supernode". Figure

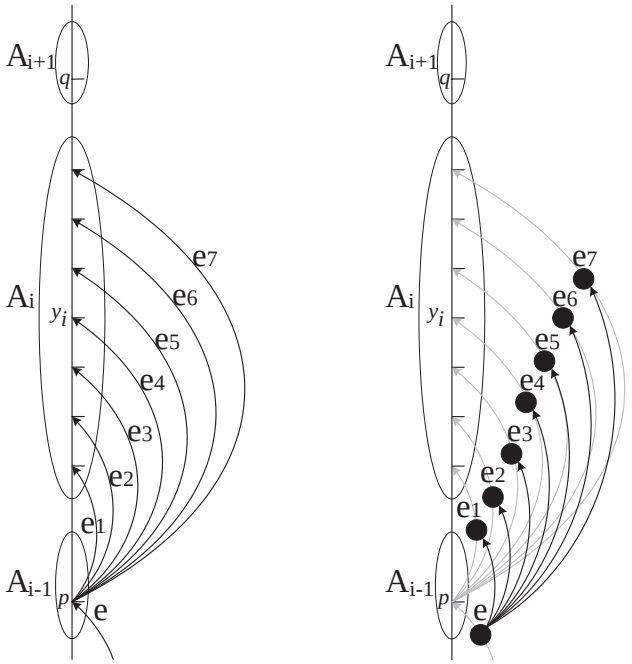


Figure 7.7 (a) FSP graph of a CVEP problem. p is a feasible position of A_{i-1}. (b) SP graph of the CVEP problem.

7.6 illustrates an example. A_i includes 12 feasible positions. When clustering 3 nodes as a "supernode", there are only 4 "supernodes". Accordingly, the size of SP graph can be greatly reduced.

7.5.2 Edge omitting

The construction of SP graph $\bar{G}$ is based on FSP graph G. During the transformation from FSP graph to SP graph, if we know that some edges will not appear in the final solution, then these edges can be omitted in SP graph. Therefore, the target of this optimization strategy is to identify this kind of edges.

Suppose a path $P = (A_1, ..., A_n)$ is the input of a CVEP problem, where $A_i (i =$

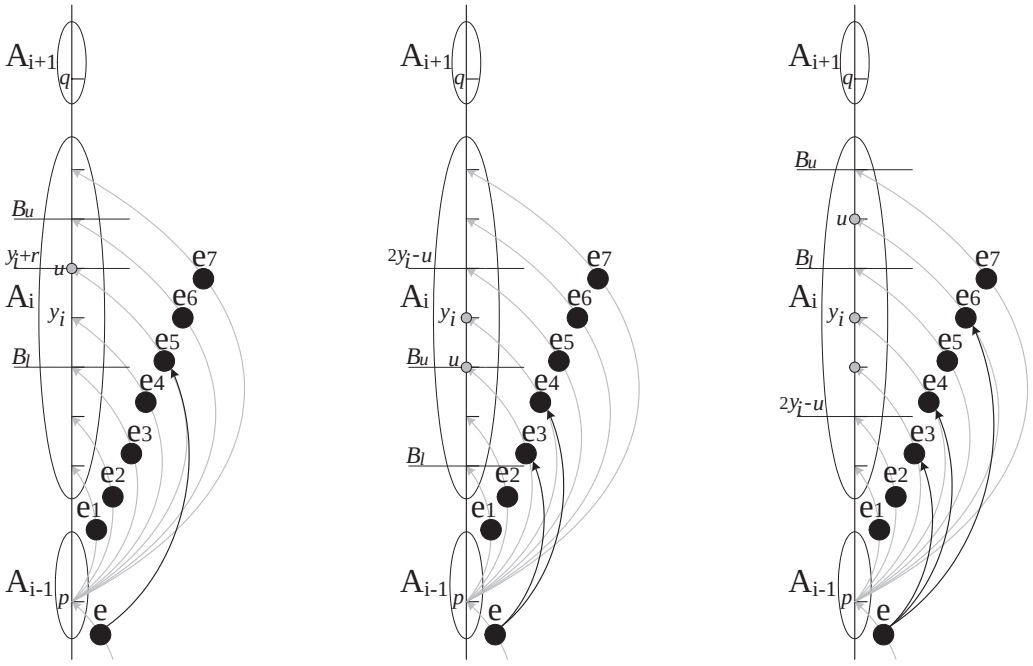
$1, ..., n)$ is a wire segment. Let $A_i = (x_1^i, x_2^i, y_i, w_i, c_i, d_i)$, and its FP-range be $[V_i, U_i]$. For convenience, we call a position is a feasible position of A_i if it is not occupied by any power rail, and its y-coordinate falls in $[V_i, U_i] \cap [y_i - d_i, y_i + d_i]$. For a feasible position p of A_{i-1}, suppose there is one edge e connecting to p either from s or a feasible position of A_{i-2} as illustrated in Figure 7.7 (a). In (a), A_i includes 7 feasible positions. p is a feasible position of A_{i-1}. e connects to p and p connects to all feasible positions of A_i. Based on this FSP graph, the corresponding SP graph is Figure 7.7 (b), assuming (e, e_i) $(i = 1, ..., 7)$ satisfies the crosstalk constraint. However, in some cases, some of these edges may not be needed.

Suppose q is the lowest feasible position of A_{i+1}. Let $B_u = \min\{q - 2s - w_i - w_{i+1}, q - D - w_i - w_{i+1}\}$, where $2s$ is the minimum spacing between two segments and if the distance of two segments is larger D, there is no crosstalk between the two segments. Also let $B_l = \max\{p + 2s + w_i + w_{i-1}, p + D + w_i + w_{i+1}\}$. We have the following cases.

Case 1 $B_l \leq y_i \leq B_u$.

Let $r = \min\{y_i - B_l, B_u - y_i\}$. Start from y_i, and search within the range $[y_i - r, y_i + r]$. If y_i is occupied by power rails, then check $y_i - 1$, $y_i + 1$, $y_i - 2$, $y_i + 2$ ... until a feasible position u is found or it is out of the range. If u is found, then only one edge is needed in the SP graph, i.e., connecting the two nodes corresponding to e and (p, u) in the FSP graph. In Figure 7.8 (Case 1), u is the closest feasible position to y_i in the range $[y_i - r, y_i + r]$. Only one edge (e, e_5) is needed in SP graph.

Consider other feasible positions v of A_i. Given an optimal solution S of a CVEP



(Case 1) (Case 2) (Case 3)

Figure 7.8 (Case 1) u is the closest feasible position to y_i. Only one edge is needed. (Case 2) y_i is the only feasible position in $(B_u, 2y_i - u)$. Two edges are added. (Case 3) There are two feasible positions in $(2y_i - u, B_l)$. Three edges are added.

problem, suppose positions p, v and w (w is a feasible position of A_{i+1}) are selected for A_{i-1}, A_i and A_{i+1} respectively. Then p, u and w must also be feasible positions of the three wire segments since the capacitive crosstalk of (p, u) and (u, w) is zero. However, u is the closest feasible position to y_i and it has the least deviation among all feasible positions of A_i. Therefore, a solution with p, u and w as the positions of A_{i-1}, A_i and A_{i+1} should have less deviation. But this contradicts that S is an optimal solution.

Case 2 $B_l \leq B_u \leq y_i$

Start from B_u, and search within the range $[B_l, B_u]$. If B_u is occupied by power rails, then check $B_u - 1$, $B_u - 2$ ... until a feasible position u is found or it is out of the range. If u is found, then add edges $(e, (p, u))$ and $(e, (p, \bar{u}))$ where $\bar{u} \in (B_u, 2y_i - u)$. As illustrated in

Figure 7.8 (Case 2), u is a feasible position in $[B_l, B_u]$, and y_i is the only feasible position in $(B_u, 2y_i - u)$. Therefore, only two edges (e, e_3) and (e, e_4) are added in SP graph.

As to other feasible positions v of A_i, it must be outside the range $[u, 2y_i - u)$. If p, v and w (w is a feasible position of A_{i+1}) are selected for wire segments A_{i-1}, A_i, and A_{i+1}, respectively, in a solution S, then there must exist a solution $\bar{S}$ with less total deviation. In $\bar{S}$, the positions of all segments are the same as those in S except that A_i is placed at u instead of v.

Case 3 $y_i \leq B_l \leq B_u$

Start from B_l, and search within the range $[B_l, B_u]$. If B_l is occupied by power rails, then check $B_l + 1$, $B_l + 2$ ... until a feasible position u is found or it is out of the range. If u is found, then add edges $(e, (p, u))$ and $(e, (p, \bar{u}))$ where $\bar{u} \in (2y_i - u, B_l)$. As shown in Figure 7.8 (Case 3), u is a feasible position in $[B_l, B_u]$, and there are two feasible positions in $(2y_i - u, B_l)$. Therefore, three edges (e, e_3), (e, e_4) and (e, e_6) are added in SP graph.

If the conditions in the above three cases are not satisfied, then just connect nodes in the original way.

7.6 Experimental Results

Our algorithms were implemented in C++ on PC (733MHz) with 128MB memory. We tested CVE algorithms for four test files in Table 7.1. These circuits were obtained from industry files. For all of the test circuits, the allowable derivation of each signal wire segment is bounded as 2% of the height of the ECO region area. After applying the FCVE

Table 7.1 Test files of CVE problem.

File	N3	S6	M8	F10
ECO Region Area (um^2)	(4908.92, 3295.52)	(3295.52, 4908.92)	(10872.90, 4799.54)	(4799.54, 10872.90)
Signal Segments	1601	2098	1266	726
Power Rail Segments	166	1128	631	747
Sensitive Segments	1439	1868	1085	683
Crosstalk Violation Segments	406	296	177	227
Allowable Deviation	2%	2%	2%	2%
Node Clustering for CVE	10	9	30	60

algorithm, we can find clean routing solutions for all four files, and the max deviations are much smaller than the given bound. Then based on the output of the FCVE algorithm, we use SCVE to further improve the total deviation. The test results in Table 7.2 show that SCVE can greatly reduce the total deviation, for example, the total deviation is reduced to less than 5% of the original total deviation for both N3 and S6.

Table 7.2 Test results of CVE problem.

File		N3	S6	M8	F10
Max Deviation		0.12%	0.01%	0.28%	0.85%
Crosstalk Violation Segments		0	0	0	0
Time (second)	FCVE	3	2	2	1
	SCVE	5	1	9	42
Total Deviation	FCVE (um)	4533.54	768.15	11713.80	51930.80
	SCVE (um)	215.32	23.28	633.90	6820.99
	FCVE/SCVE	4.75%	3.03%	5.41%	13.13%

Table 7.3 Optimization for test file N3.

Node Clustering	Total Deviation	Time (second)	
		NEO	EO
2	160.23	251	149
4	176.56	49	32
6	198.77	22	14
8	209.17	12	8
10	215.32	8	5

Moreover, we tested the optimization strategies on the test file N3. Table 7.3 shows the test results of different granularity of node clustering. When more nodes are clustered as a "supernode", the running time is much shorter although the total deviation is a little larger. At the same time, the experimental results show that edge omitting optimization strategy is also very effective such that the running time can be shortened by $1/3$. NEO means no edge omitting is adopted; while EO refers to edge omitting.

7.7 Conclusion

In this chapter, we present a two-stage algorithm to solve the CVE (Crosstalk Violation Elimination) problem. The first stage processes signal wire segments one by one and tries to find a clean routing solution. Then efforts are made in the second stage to minimize the total deviation. Furthermore, in order to facilitate the process of huge problems, we propose efficient optimization strategies to speed up the execution. Experimental results demonstrate the efficiency and effectiveness of our approach.

CHAPTER 8

CONCLUSION

8.1 Summary

Physical design plays an important role of connecting front-end design and back-end design in chip development. In this thesis, we present various algorithms for problems in VLSI physical design.

In Chapter 2, we propose bus-driven floorplanning that considers floorplanning and bus planning simultaneously. An efficient evaluation algorithm is developed to transform a sequence pair representation to a BDF solution which is a placement of all circuit blocks such that each bus can be realized as a rectangular strip (horizontal or vertical) going through all the blocks connected by the bus.

In Chapter 3, we address the wire planning problem with bounded over-the-block constraints. We present two exact polynomial-time algorithms. Both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists. One requires less memory, while the other is faster when processing a large number of nets.

In Chapter 4, we present the first polynomial-time optimal algorithm for simultaneous pin assignment and routing in multilayer for all two-pin nets between a source block and

all other blocks. Our algorithm is applicable for both global routing and detailed routing with arbitrary routing obstacles on multiple layers, and guarantees a pin-assignment and routing solution with minimum total cost $\alpha \cdot W + \beta \cdot V$ where W is the total wire length and V is the number of vias. This algorithm matches well with ECO situations and can be used to improve any routing solution.

In Chapter 5, we propose a polynomial-time algorithm for simultaneous pin assignment and buffer planning for all 2-pin nets between a source macro block and all other blocks such that each net satisfies the lower and upper bounds of connection intervals as well as minimizing the total cost $\alpha \cdot W + \beta \cdot R$ where W is the total wire length and R is the number of buffers. By applying this algorithm iteratively (i.e., each time selecting one block as the source block), it provides a polynomial-time algorithm for pin assignment and buffer planning for nets among multiple macro blocks.

In Chapter 6, we present two polynomial-time algorithms to resolve the overlaps between power rails and signal wires on the top layer as well as satisfying other constrains. Both algorithms guarantee to find a feasible solution as long as one exists. One is faster, while the other makes effort to reduce total deviation. According to different application requirements, users can choose an appropriate one.

In Chapter 7, we propose a two-stage algorithm to solve the CVE problem. The target is to find a new routing solution without crosstalk violations under certain constraints which help to keep the new design close to the original one. Furthermore, in order to handle very large problems, we propose efficient optimization strategies to speed up the execution. This

algorithm can also be used to eliminate crosstalk violations in the output of the above ECO wire legalization problem.

8.2 Future Research

We now address some possible directions for the future research.

Advances in fabrication technology allow exponential growth in the number of transistors integrated on a die and keep modest increase in the cost of manufacturing process. However, the increase in design cost cannot parallel with the flat increase rate in manufacturing cost. Proliferation, a process of making a new design by modifying an existing product, such as product upgrade for speed push, saves design efforts from designing a totally new chip and produces main revenue from the initial design. For high-end and high-volume products, one good option of further improving chip performance is to add extra metal layers based on an existing design after all easy and quick circuit fixes and process tricks are already applied. Since the decision window is very limited, the utilization of the newly added metal layer is a challenging task for design re-optimization. For example, how to migrate wire segments to the new layer so that the coupling capacitance can be minimized. Also how to adjust segment positions when the two layers have layer-dependent design spacing rules.

Another exciting new direction of research is to investigate non-Manhattan design which is prompted by X-architecture. Compared with the Manhattan, the wire length can be reduced by more than 20%, the via deduction can be more than 30% and the chip size

shrunk can be 10% [48]. However, to take full advantages of X-architecture, it is necessary to develop X-aware floorplan and placement as well as mixed Manhattan-Diagonal routing. Research in this direction should be very valuable.

REFERENCES

[1] S. M. Sait and H. Youssef, *VLSI Physical Design Automation - Theory and Practice*, McGraw-Hill Book Company, 1995.

[2] F. Y. Young, C. N. Chu and M. L. Ho, "A unified method to handle different kinds of placement constraints in floorplan design," in *Proceedings of the 15th International Conference on VLSI Design*, 2002, pp. 661-667.

[3] X. Tang and D. F. Wong, "Floorplanning with alignment and performance constraints," in *Proceedings of ACM/IEEE Design Automation Conference*, 2002, pp. 848-853.

[4] R. Liu, X. Hong, S. Dong, Y. Cai, and J. Gu, "VLSI/PCB placement with predefined coordinate alignment constraint based on sequence pair," in *Proceedings of the 4th International Conference on ASIC*, 2001, pp. 167-170.

[5] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Bus-based integrated floorplanning," in *IEEE International Symposium on Circuits and Systems*, 2002, pp. 875-878.

[6] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Integrated floorplanning with buffer/channel insertion for bus-based microprocessor designs," in *Proceedings of International Symposium on Physical Design*, 2002, pp. 56-61.

[7] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair,” *IEEE Trans. on Computer-Aided Design*, vol. 15:12, pp. 1518-1524, 1996.

[8] X. Tang and D. F. Wong, “FAST-SP: a fast algorithm for block placement based on sequence pair,” in *ASP-DAC*, 2001, pp. 521-526.

[9] X. Tang, R. Tian, and D. F. Wong, “Fast evaluation of sequence pair in block placement by longest common subsequence computation,” in *DATE-00*, 2000, pp. 106-111.

[10] L. D. Huang, M. H. Lai, D. F. Wong, and Y. X. Gao, “Maze routing with buffer insertion under transition time constraints,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22-1, pp. 91-95, Jan 2003.

[11] J. Lillis, C. K. Cheng, and T. T. Lin, “Optimal and efficient buffer insertion and wire sizing,” in *CICC*, 1995, pp. 259-262.

[12] L.P.P.P. van Ginneken, “Buffer placement in distributed RC-tree networks for minimal elmore delay,” in *ISCAS*, 1990, pp. 865-868.

[13] H. Zhou, D. F. Wong, I-M. Liu, and A. Aziz, “Simultaneous routing and buffer insertion with restrictions on buffer locations,” *IEEE Transaction on Computer-Aided Design*, 2000.

[14] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, Prentice Hall, 1993.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1992.

[16] H. N. Brady, "An approach to topological pin assignment," *IEEE Transaction on Computer-Aided Design*, vol. CAD-3, pp. 250-255, 1984.

[17] N. L. Koren, "Pin assignment in automated printed circuit board design," in *Proceedings of ACM/IEEE Design Automation Conference*, 1972, pp. 72-79.

[18] L. Mory-Rauch, "Pin assignment on a printed circuit board," in *Proceedings of ACM/IEEE Design Automation Conference*, 1978, pp. 70-73.

[19] X. Yao, M. Yamada, and C. L. Liu, "A new approach to the pin assignment problem," in *Proceedings of ACM/IEEE Design Automation Conference*, 1988, pp. 566-572.

[20] S. G. Choi and C. M. Kyung, "Three-step pin assignment algorithm for building block layout," *Electron. Lett.*, vol. 28, no. 20, pp. 1882-1884, 1992.

[21] J. Cong, "Pin assignment with global routing for general cell designs," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 1401-1412, 1991.

[22] T. Koide, S. Wakabayashi, and N. Yoshida, "An integrated approach to pin assignment and global routing for VLSI building-block layout," in *Proceedings of European Conference on Design Automation with the European Event in ASIC Design*, 1993, pp. 24-28.

[23] L. E. Liu and C. Sechen, “Multilayer pin assignment for macro cell circuits,” *IEEE Transaction on Computer-Aided Design*, vol. 18, pp. 1452-1461, 1999.

[24] L. Y. Wang, Y. T. Lai, and B. D. Liu, “Simultaneous pin assignment and global wiring for custom VLSI design,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, 1991, vol. 4, pp. 2128-2131.

[25] C. Albrecht, “Provably good global routing by a new approximation algorithm for multicommodity flow,” in *ACM International Symposium on Physical Design*, 2000, pp. 19-25.

[26] C. Albrecht, A. B. Kahng, I. Mandoiu, and A. Zelikovsky, “Floorplan evaluation with timing-driven global wireplanning, pin assignment, and buffer/wire sizing,” in *IEEE ASP-DAC*, 2002, pp. 580-587.

[27] R. C. Carden and C. K. Cheng, “A global router using an efficient approximate multicommodity multiterminal flow algorithm,” in *Proceedings of ACM/IEEE Design Automation Conference*, 1991, pp. 316-321.

[28] J. D. Cho and M. Sarrafzadeh, “Four-bend top-down global routing,” *IEEE Transaction on Computer-Aided Design*, vol. 17, pp. 793-802, 1998.

[29] J. Huang, X. L. Hong, C. K. Cheng, and E. S. Kuh, “An efficient timing-driven global routing algorithm,” in *Proceedings of ACM/IEEE Design Automation Cconference*, 1993, pp. 596-600.

[30] G. Meixner and U. Lauther, “A new global router based on a flow model and linear assignment,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1990, pp. 44-47.

[31] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan, “Finding minimum-cost flows by double scaling,” *Mathematical Programming*, pp. 243-266, 1992.

[32] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, 1990.

[33] R. Otten, “Global wires harmful?” in *Proceedings of International Symposium on Physical Design*, 1998, pp. 104-109.

[34] J. Cong, “Challenges and opportunities for design innovations in nanometer technologies,” SRC Working Papers, Dec 1997, http://www.src.org/prg_mgmt/frontier.dgw.

[35] J. Cong, T. Kong, and D. Z. Pan, “Buffer block planning for interconnect driven floorplanning,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1999, pp. 358-363.

[36] X. Tang and D. F. Wong, “Planning buffer locations by network flows,” in *Proceedings of International Symposium Physical Design*, 2000, pp. 180-185.

[37] P. Sarkar, V. Sundararaman, and C. K. Koh, “Routability-Driven repeater block planning for interconnect-centric floorplanning,” in *Proceedings of International Symposium on Physical Design*, 2000.

[38] F. F. Dragan, A. B. Kahng, I. I. Mandoiu, S. Muddu, and A. Zelikovsky, “Provably good global buffering using an available buffer block plan,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2000, pp. 104-109.

[39] F. F. Dragan, A. B. Kahng, I. I. Mandoiu, S. Muddu, and A. Zelikovsky, “Provably good global buffering by multiterminal multicommodity flow approximation,” in *Proc. ASP-DAC*, 2001, pp. 120-125.

[40] B. Preas and M. Lorenzetti, *Physical design automation of VLSI systems*, Benjamin/Cummings, 1988.

[41] T. E. Dillinger, *VLSI Engineering*, Prentice Hall, 1988.

[42] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1993.

[43] F. Mo, A. Tabbara, and R. K. Brayton, “A force-directed macro-cell placer,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2000.

[44] S. Nag and K. Chaudhary, “Post-placement residual-overlap removal with minimal movement,” in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 1999.

[45] N. Quinn and M. A. Breuer, “A forced directed component placement procedure for printed circuit boards,” *IEEE Transaction on Circuits and Systems*, 1979.

[46] H. Zhou and D. F. Wong, “Global routing with crosstalk constraints,” in *Proceedings of ACM/IEEE Design Automation Conference*, 1998.

[47] T. Sakurai and K. Tamaru, “Simple formulas for two and three dimensional capacitance,” *IEEE Transaction on Electron Devices*, 1993.

[48] S. L. Teig, “The X architecture: not your father’s diagonal wiring,” in *Proceedings of the 2002 International Workshop on System-level Interconnect Prediction*, 2002, pp. 33-37.

APPENDIX

PUBLICATIONS

Journal Papers

1. H. Xiang, X. Tang, and D. F Wong, "Bus-driven floorplanning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

2. L. Huang, X. Tang, H. Xiang, D. F. Wong, and I. Liu, "A polynomial time-optimal diode insertion/routing algorithm for fixing antenna problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, No. 1, pp. 141-147, January 2004.

3. H. Xiang, X. Tang, and D. F. Wong, "Min-cost flow based algorithm for simultaneous pin assignment and routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, No. 7, pp. 870-878, July 2003.

Conference Papers

1. H. Xiang, K. Chao, and D. F. Wong, "An ECO algorithm for eliminating crosstalk violations," in *ACM/SIGDA 2004 International Symposium on Physical Design*, 2004.

2. H. Xiang, X. Tang, and D. F. Wong, “Bus-driven floorplanning,” in *IEEE/ACM International Conference on Computer Aided Design*, 2003.

3. S. Lee, H. Xiang, D. F. Wong, and R. Sun, “Wire type assignment for FPGA routing,” in *ACM International Symposium on Field-Programmable Gate Arrays*, 2003.

4. H. Xiang, K. Chao, and D. F. Wong, “ECO algorithms for removing overlaps between power rails and signal wires,” in *IEEE/ACM International Conference on Computer Aided Design*, 2002, pp. 67-74.

5. H. Xiang, X. Tang, and D. F. Wong, “An algorithm for integrated pin assignment and buffer planning,” in *ACM/IEEE Design Automation Conference, New Orleans*, 2002.

6. L. D. Huang, X. Tang, H. Xiang, D. F. Wong, and I. M. Liu, “A polynomial time optimal diode insertion/routing algorithm for fixing antenna problem,” in *Design, Automation and Test in Europe*, 2002.

7. H. Xiang, X. Tang, and D. F. Wong, “An algorithm for simultaneous pin assignment and routing,” in *IEEE/ACM International Conference on Computer Aided Design*, 2001, pp. 232-238.

8. X. Tang, R. Tian, D. F. Wong, and H. Xiang, “A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints,” in *IEEE/ACM International Conference on Computer Aided Design*, San Jose, 2001, pp. 49-56.

9. H. Xiang, X. Tang, and D. F. Wong, “Simultaneous pin assignment and routing,” in *The Tenth Workshop on Synthesis and System Integration of Mixed Technologies*, 2001.

10. L.D. Huang, X. Tang, H. Xiang, D. F. Wong, and I. M. Liu, “An exact diode insertion/routing algorithm for fixing antenna problem,” in *The Tenth Workshop on Synthesis And System Integration of Mixed Technologies*, 2001.

VITA

Hua Xiang was born in Shanghai, P.R.China, on December 26, 1973, the daughter of Xinmin Xiang and Xueying Xu. She received the B.S. and M.S. degree from the Computer Science and Technology Department of Peking University, China, in 1997 and 2000, respectively. She entered the Department of Computer Sciences of University at Texas at Austin in 2000 with MCD Fellowship. In 2002, she transferred to the department of Computer Science of University of Illinois at Urbana-Champaign. She joined Cadence Design Systems Inc. in San Jose, CA in Jan 2004.

She received many awards and honors during her undergraduate and graduate studies. She has published 3 journal papers and 10 conference papers in VLSI physical design in her Ph.D. study.